UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

Dpto. de INGENIERÍA DE SISTEMAS Y AUTOMÁTICA

INTERFAZ DE CONTROL DEL ROBOT HUMANOIDE HOAP3 PARA LA REALIZACIÓN DE TAREAS ONLINE

PROYECTO FIN DE CARRERA

INGENIERÍA TÉCNICA INDUSTRIAL: ELECTRÓNICA INDUSTRIAL

Autor: ÁLVARO DONOSO PARRADO

Tutores: PAOLO PIERRO Y DANIEL HERNANDEZ

AGRADECIMIENTOS

En primer lugar agradezco a mi familia y amigos por su apoyo durante todos estos años y sobretodo durante la elaboración de este proyecto.

Agradezco a Paolo Pierro por haberme dado la posibilidad de realizar este proyecto.

Agradezco también a Daniel Hernández por estar siempre dispuesto a ayudarme y sin cuyo tiempo, conocimientos y ayuda no hubiera sido posible sacarlo adelante.

ÍNDICE GENERAL

CAPÍTULO 1: INTRODUCCIÓN

CAPÍTULO 2: ESTADO DEL ARTE

CAPÍTULO 3: PLATAFORMA UTILIZADA

CAPIÍULO 4: INTERFAZ

ÍNDICE FIGURAS

CAPÍTULO 1: INTRODUCCIÓN

1.1 INTRODUCCIÓN

En el departamento de Ingeniería de Sistemas y Automática de la Universidad Carlos III de Madrid, se están desarrollando diversos estudios e investigaciones dentro del campo de la Robótica. El presente trabajo pretende mostrar una Interfaz de control del Robot Humanoide HOAP-3.

1.1.1 INTRODUCCIÓN A LA ROBÓTICA

Cuando hablamos de Robótica nos referimos a la rama de la tecnología que estudia la confección y el uso de los Robots.

El término Robot no es un término que haya sido concebido en el siglo XXI, sino que tiene unas raíces bastantes lejanas respecto a la actualidad. Concretamente deriva de las palabras checas "robota" y "robotnik", que significa labor y trabajador respectivamente.

La primera vez que la palabra Robot fue utilizada tal y como hoy la entendemos fue en 1920, cuando Karel Capek (1890-1938) estrenó en Praga su obra teatral "Rossum's Universal Robot".

No podemos caer en el error de pensar que los Robots son sólo cosa de nuestra sociedad. Esto no es así; si nos remontamos inclusive a la antigua cultura griega nos encontramos con lo que ellos llamaban "autómatas"; máquinas que efectuaban movimientos de manera independiente; famosos son por todo esto los mecanismos de Heron de Alejandría. (S.I d.c). Avanzando a lo largo del tiempo nos encontramos con la cultura árabe, famosa a su vez por crear un sistema dispensador de agua para la realeza.

Aún con todo, las auténticas máquinas autónomas no aparecieron hasta el siglo XX.

1.1.2 TIPOS DE ROBOTS

Los Robots han sido clasificados aplicando diversos criterios: generación, nivel de inteligencia, capacidad de control o lenguaje de programación. La generación viene vinculada al orden de desarrollo histórico de la Robótica. Algunos opinan que podemos hablar de cinco generaciones. Aquí solo hablaremos de cuatro de ellas ya que consideramos que la quinta es una utopía (Robots con inteligencia artificial, capaces de resolver problemas y tomar sus propias decisiones).

La primera generación se caracteriza porque su sistema de control está fundamentado en "paradas fijas mecánicamente".Son sistemas mecánicos multifunciones con un sistema de control elemental, bien manual, bien automático, de secuencia fija o variable. Como ejemplo de esta etapa están los mecanismos de relojería que mueven las cajas musicales.

En cuanto a los Robots de segunda generación podemos considerar a aquellos a los que se les asigna una tarea según una serie de instrucciones programadas previamente y las ejecutan de forma secuencial. Una de las características de los ordenadores de esta generación es que no tienen en cuenta las características del entorno, si este varía o no varía. Los Robots de segunda generación se utilizaban en la industria automotriz y destacaban por sus grandes dimensiones.

La característica anteriormente mencionada es la que lo diferencia de los Robots de tercera generación. Estos contienen un sistema de control de lazo cerrado; por lo tanto, mediante sensores adquieren información sobre el entorno que le rodea y se adaptan a dicha situación; es decir, son reprogramables. Podemos hablar de que en esta generación aparecen los primeros lenguajes de programación para escribir los programas de control.

Y por último los Robots de cuarta generación son Robots que se caracterizan por su gran nivel de inteligencia. El mayor nivel de inteligencia se debe a las mejoras de sus extensiones sensoriales. Este progreso ha hecho que el Robot sea capaz de enviar información a la computadora de control acerca del estado del proceso.

Como ya hemos dicho hay diversas maneras de clasificar a los Robots, esta vez los clasificaremos por su arquitectura.

Empezaremos por los poliarticulados. Bajo este grupo están los Robots de diversa forma y configuración que se caracterizan todos ellos por ser sedentarios (no pueden realizar desplazamientos, pueden mover alguna de sus partes, las más frecuente las extremidades, con algunos grados de libertad, dentro de su espacio de trabajo según uno o más sistemas de coordenadas)

En esta clasificación se encuentran los Robots industriales y los Robots cartesianos, como el de la figura 1.

FIGURA 1: Robot poliarticulado

Los móviles, como el mostrado en la figura 2, son Robots, que como su propio nombre indica tienen gran capacidad de desplazamiento debido a un sistema locomotor de tipo rodante. El desplazamiento es ordenado por telemando, o por un

complejo sistema de sensores que le envía la información percibida en su entorno. La función de estos Robots es transportar piezas de un lugar a otro de la cadena de fabricación. Guiados mediante pistas materializadas a través de la radiación electromagnética de circuitos empotrados en el suelo, o a través de bandas detectadas fotoeléctricamente, pueden incluso llegar a sortear obstáculos y están dotados de un nivel relativamente elevado de inteligencia.

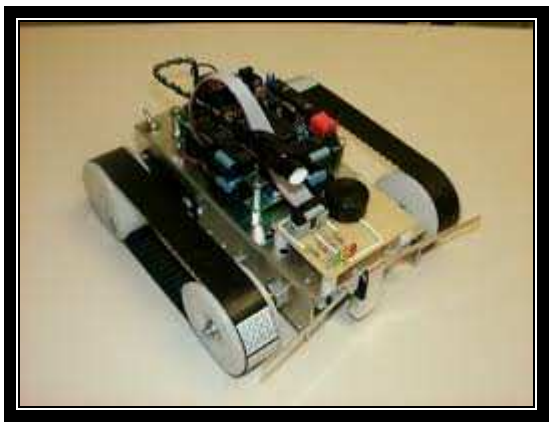

FIGURA 2: Robot móvil

En tercer lugar vamos a analizar aquellos Robots a los que llamamos Androides, como el mostrado en la figura 3.

Los Androides son aquellos que intentan de una manera parcial o total reproducir la forma de actuación del ser humano. Estos Robots carecen de utilidad práctica ya que están muy poco desarrollados y por lo general están destinados, fundamentalmente, al estudio y a la experimentación. Uno de los aspectos más complejos de estos Robots, y sobre el que se centra la mayoría de los trabajos, es el de la locomoción bípeda. En este caso, el principal problema es controlar dinámica y coordinadamente en el tiempo real el proceso y mantener simultáneamente el equilibrio del Robot.

FIGURA 3: Robot androide

Se puede hablar a su vez de los Robots zoomórficos, como el de la figura 4, a los que podríamos incluir en los Robots androides. Una de sus características principales es que mediante su sistema de locomoción imitan a ciertos seres vivos. A los Robots zoomórficos se les puede agrupar a su vez: en caminadores y no caminadores .El grupo de los Robots zoomórficos no caminadores es el que está menos evolucionado. Los Robots zoomórficos caminadores multípedos son muy numerosos y están siendo experimentados en diversos laboratorios con vistas al desarrollo posterior de verdaderos vehículos terrenos, capaces de evolucionar en superficies muy accidentadas. Las aplicaciones de estos Robots serán interesantes en el campo de la exploración espacial y en el estudio de los volcanes.

FIGURA 4: Robot zoomórfico

1.1.3 ARQUITECTURAS ROBÓTICAS

Todos y cada uno de los Robots se construyen sobre ciertas arquitecturas; para que se construya un buen Robot, éste debe basarse en un control software que indica al Robot que tarea realizar en cada momento, que acción o que movimiento. Según Javier de Lope (2006) toda arquitectura de control debe tener unas características básicas, de no ser así; no se asegura el funcionamiento del Robot al cien por cien. Éstas son:

- Capacidad de abordar múltiples objetivos de forma simultánea.
- Capacidad de integración de la información de múltiples sensores de diferente procedencia.
- Robustez ante fallos de elementos del sistema.
- Adaptación ante nuevos entornos.

- Capacidad de extensión y modificación a lo largo de su vida.

- Capacidad para considerar posibilidades y valorar consecuencias.

- Interacción con el entorno para percibir cambios en él y responder adecuadamente.

La elección de una u otra arquitectura no sigue ningún parámetro, ya que no hay ninguna normativa que indique cuál es la arquitectura óptima para cada tipo de Robot, depende de muchos factores que deben ser elegidos por aquella persona que desarrolle el Robot. Aún así hay características comunes que nos ayudan a declinarnos entre una arquitectura u otra.

Hay tres elementos conocidos universalmente y relacionados entre sí en Robótica; detección, planificación y actuación. El Robot, podemos decir, que se divide en tres partes que coinciden con estos tres elementos. En la detección, el Robot, mediante los sensores, recoge la información que se da en su entorno. Tras esto se encarga de decidir cuál será la acción que llevará a cabo (planificación) y por último, en la parte de ejecución, realiza la acción.

Además hay que tener en cuenta la manera en que es procesada la información recibida a través de los sensores; es decir la reacción que se ofrece cuando se recibe información del exterior por medio de sensores.

Teniendo en cuenta lo expuesto anteriormente, existen tres paradigmas en los que se basan las arquitecturas actuales: el paradigma jerárquico, el paradigma reactivo, y un paradigma híbrido (mezcla entre los dos anteriores).

Para que estos paradigmas sean efectivos en la vida real, las arquitecturas Robóticas se basaran en éstos.

1.1.4 ROBOTS HUMANOIDES

A continuación vamos ha hacer un análisis más profundo de los Robots Humanoides ya que como he avanzado antes el presente trabajo pretende mostrar una Interfaz de Control de movimiento del Robot Humanoide HOAP-3.

Un Robot Humanoide destaca principalmente por tener una apariencia similar a la del cuerpo humano. Esta característica es lo que permite la interacción con los humanos a través de herramientas o ambientes.
El diseño de los Robots Humanoides por lo general siempre es el mismo: Un torso con cabeza, dos brazos y dos piernas. En algún caso, se puede plantear un modelo de Robot Humanoide que sólo conste de una parte del cuerpo, como puede ser de cintura para arriba o incluso únicamente un brazo o una pierna.
Podemos decir, a ciencia cierta, que un Robot Humanoide es un Robot autónomo ya que puede adaptarse a los cambios que se dan en su entorno y así poder seguir en sus funciones y alcanzar la meta. Esto es una de esas características que diferencian a los Robots Humanoides de otros Robots. En este contexto podemos citar una serie de capacidades que hacen a los Robots Humanoides únicos; estas son:

1- Auto-mantenimiento (como la recarga de sí mismo)

2- El aprendizaje autónomo (aprender o adquirir nuevas capacidades sin ayuda exterior, las estrategias de ajuste basado en el entorno y adaptarse a nuevas situaciones)

3-Evitar situaciones perjudiciales para las personas.

4- Interacción segura con los seres humanos y el medio ambiente.

No nos podemos olvidar que aunque los Robots Humanoides tienen numerosas características que les hacen resaltar sobre los demás, son Robots y por consiguiente tienen los siguientes componentes básicos también:

1) Perfección
2) Planificación
3) Control

Por todo lo expuesto anteriormente, podemos concluir diciendo que los Robots Humanoides son de los Robots más complejos debido a su carácter autómata y por la gran semejanza que demuestran en la estructura y el comportamiento humano.

1.2 MOTIVACIÓN

El hombre trata constantemente de hacer su vida más cómoda y segura , es decir, trata de conseguir, con la mínima intervención por parte del individuo y con una mayor eficiencia, la realización de aquellas tareas básicas que se llevan a cabo normalmente o incluso aquellas que puedan resultar peligrosas, aburridas o, porqué no, imposibles de realizar por el hombre.

El Control Automático y la Robótica son las respuesta a estas inquietudes .El Control Automático porque nos permite automatizar los procesos y de esta forma minimizar el tiempo que el hombre emplea en su supervisión. La Robótica, porque combinada con el Control Automático, nos facilita el acceso a tareas que de otra forma sólo se podrían llevar a cabo invirtiendo en riesgo y esfuerzo humano.

Sin embargo la presencia del hombre sigue siendo necesaria, puesto que es el hombre quién ha de diseñar el control, es el hombre quien ha de diseñar el dispositivo y, finalmente, es el hombre quién ha de determinar que trabajo ha de

realizar el Robot, además de cómo y cuándo quiere que se lleve a cabo. Para ello es necesario que exista un canal de comunicación entre el individuo y el sistema, lo que se conoce como Interfaz de Usuario. Hay múltiples tipos de Interfaces de Usuario y su complejidad dependerá del tipo de usuario al que estén dirigidas y de las actividades que se pretenden realizar. Éste es el punto de partida del proyecto.

.

1.3 OBJETIVOS

El objetivo principal de este proyecto es la implementación de una Interfaz de Control del Robot Humanoide HOAP-3 para la realización de tareas online.

La aplicación desarrollada deberá permitirnos conseguir el establecimiento de comunicación bidireccional con el dispositivo móvil de forma que tanto desde el PC como desde el controlador del Robot se puedan mandar y recibir datos.

En primer lugar se desarrollará una Interfaz Gráfica de Usuario en la plataforma Flex, atendiendo a los controles y comandos requeridos por el Robot, en el que se dispondrá de la imagen que éste está visualizando en tiempo real, una serie de comandos con los que podremos controlar la dirección que toma e incluso la orientación y rotación de su cabeza. Una vez desarrollada la Interfaz, nuestro siguiente objetivo es establecer una comunicación rápida y robusta entre los distintos dispositivos para hacer efectivo el control del Robot. Por último, se probarán dichas tareas en el propio Robot para comprobar el funcionamiento.

1.4 ORGANIZACIÓN DEL DOCUMENTO

A continuación se hace referencia a los capítulos que componen la memoria del proyecto incluyendo un breve comentario explicando su contenido:

-A lo largo del capitulo 2, Estado del Arte, se hará mención de otras Interfaces Graficas de Usuario que nos sirvan como referencia y presentaremos una serie de herramientas con las que se puede desarrollar una Interfaz.

-En el capítulo 3, Plataforma Utilizada, se expondrá y profundizará en la plataforma seleccionada además de un análisis de la misma.

-En el capítulo 4, Interfaz, se analizará el desarrollo y construcción de nuestra Interfaz describiendo cada una de las partes de la misma.

-En el capítulo 5, Implementación en el HOAP, describiremos a nuestro Robot, cómo se ejecuta la comunicación con él y desarrollaremos una prueba de ejecución.

-En el capítulo 6, Conclusiones, se extraerán las conclusiones pertinentes

-Por último, en el capítulo 7, Bibliografía, se detallarán las distintas fuentes de información utilizadas en éste proyecto.

CAPÍTULO 2: ESTADO DEL ARTE

2.1 OTROS INTERFACES

El desarrollo de nuevas Interfaces es actualmente un campo de investigación activo.

A la hora de hacer mención de otras Interfaces de Usuario cabe destacar el proyecto ROBOT@CWE en el que se desarrolló una Interfaz de Control de Robot Humanoide HOAP-3, cuya apariencia es la mostrada en la figura 5.

La HRI pretende ofrecer un manejo sencillo e intuitivo de forma que sea fácil de usar para aquellos que no estén familiarizados con el robot humanoide HOAP-3.

Sus principales funciones son:

-Conexión con el Robot vía TCP/IP.

-Visualización fluida del vídeo procedente de la cámara del Robot.

-Control del movimiento y velocidad del Robot.

-Control del movimiento de la cabeza.

-Enviar comandos de alto nivel al Robot.

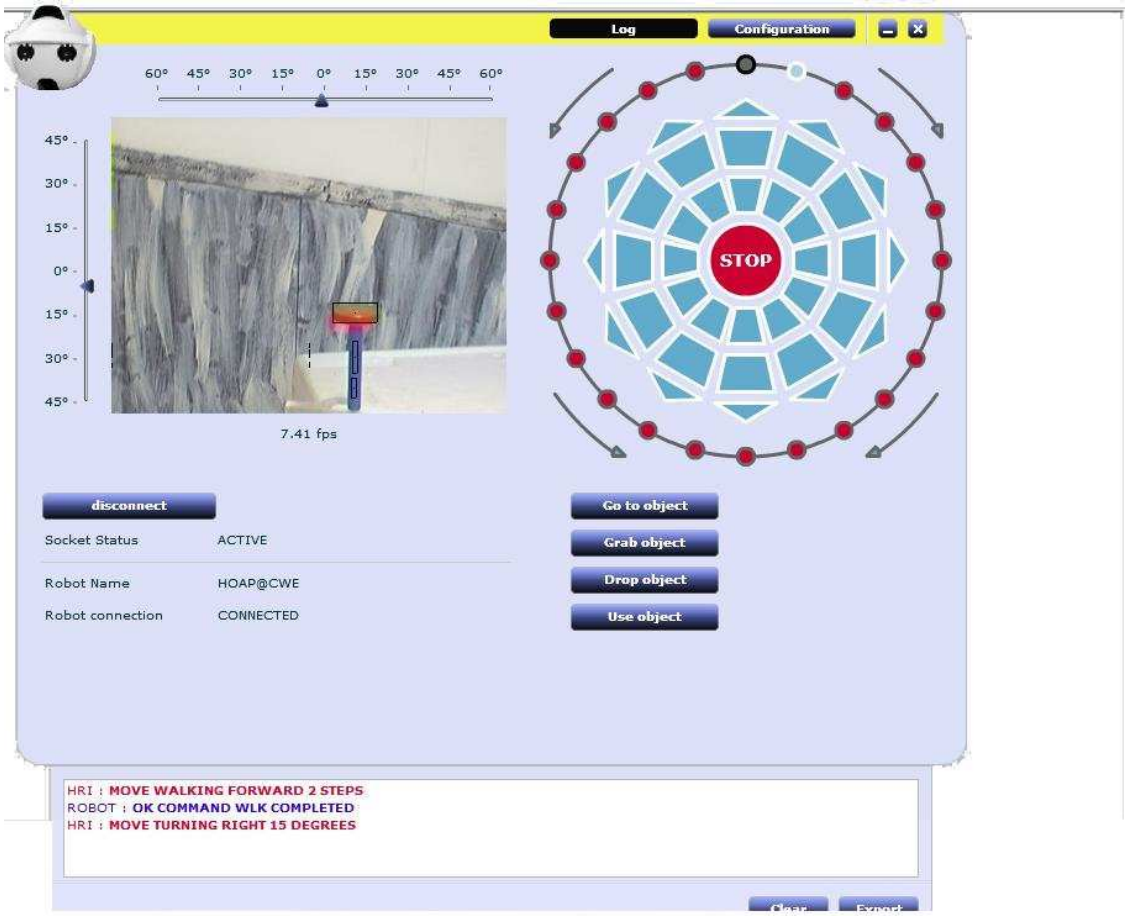

FIGURA 5: Interfaz ROBOT@CWE

Nuestro proyecto se ha desarrollado tomando como referencia éste de cuyo código no se disponía

2.2 OTRAS HERRAMIENTAS PARA DESARROLLO DE INTERFACES

Para el desarrollo de Interfaces de Usuario existen multitud de plataformas, algunas de ellas son:

2.2.1 UIML

User Interface Markrup Language (UIML) es un lenguaje de especificación de Interfaces de Usuario que permite obtener definiciones declarativas de Interfaces de Usuario independientes de la plataforma, tipo de interacción y del lenguaje de programación. Para escribir una Interfaz en UIML se debe realizar, por un lado una definición de la Interfaz genérica y por otro lado un documento UIML que representa el estilo de presentación apropiado para el dispositivo en el cuál la Interfaz se va a ejecutar. De éste modo, una misma aplicación sólo necesitará un documento UIML de especificación para cualquier dispositivo y un documento de estilo para cada dispositivo.

2.2.2. OPENLASZLO

OpenLaszlo es un entorno de desarrollo *opensource* para crear Aplicaciones Ricas en Internet (RIA), consistente en tres aplicaciones:

1- Compiler, para compilar el código fuente en formato Flash.

2- Runtime, Framework que incluye la Interfaz de Usuario y componentes para ser usados

3-Servlet, que admite otros tipos de archivos, SOAP y XML-RPC.

2.2.3.Qt

Qt se puede definir como una herramienta software utilizada para la creación de Interfaces Gráficas de Usuario, (GUI), por la empresa Trolltech.
Consiste en un conjunto de librerías multiplataforma para el desarrollo de aplicaciones GUI escritas en C++, lo que indica que está orientada a objetos.
Qt permite crear aplicaciones para diversas plataformas, como son: Windows desde Windos 95 hasta XP, Mac OS X, Linux Solaris, HP-UX y otras versiones de Unix con X11. El desarrollo de una aplicación o GUI para una determinada plataforma no implica cambios importantes si se quiere utilizar en otra plataforma diferente.
Sus características más relevantes son:

1-Dispone de una librería para Interfaces Gráficos.

2-Está orientada a objetos. Usa el lenguaje C++.

3-Se basa en el concepto de Widgets (objetos de una paleta), Señales, Slots y Eventos.

4-Qt también dispone de otras funciones.

5-Librerías básicas.

6-Entrada/Salida, Manejo de Red, XML.

7-Interfaces con Bases de Datos: MySQL, Oracle.

8-Plugings (interfaces con Bases de Datos):

2.2.4,XML-based User-interface Language(XUL)

XUL es un lenguaje basado en XML utilizado para describir y crear Interfaces de Usuario, que ha sido diseñado para brindar la portabilidad de las mismas, por lo que permite desarrollar aplicaciones multi-plataforma sofisticadas o complejas sin necesidad de herramientas especiales. Inicialmente XUL fue creado para desarrollar los productos de Mozilla (navegador y cliente de e-mail) de una forma más rápida y fácil. Al ser un lenguaje basado en XML, contiene todas las características disponibles para XML y sus mismas ventajas. Una Interfaz XUL se define mediante la especificación de tres grupos de componentes distintos:

1-*Content:* Aquí se encuentran los documentos XUL, que definen el diseño de la Interfaz.

2-*Skin:* Contiene las hojas de estilos (CSS) y las imágenes, las cuáles definen la apariencia de la interfaz.

3-*Locale:* Los documentos DTD se encuentran aquí, estos documentos facilitan la localización de páginas XUL.

2.2.5 AJAX

Una de las opciones mas fáciles de ejecución en las que se basan las aplicaciones RIA actuales es Ajax, que es el acrónimo de *Asynchronous JavaScript and XML*(JavaScript y XML Asíncronos). AJAX se basa en herramientas que ya son

familiares a los desarrolladores Web: HTML, DHTML y JavaScript. La idea fundamental en la que se basa AJAX es utilizar JavaScript para actualizar la página sin tener que volver a cargarla.

Un programa JavaScript que se ejecuta en un navegador puede introducir nuevos datos en la página o cambiar su estructura manipulando HTML DOM (*HTML Document Object Model,* Modelo de Documentos Objeto HTML) sin tener que volver a cargar una página nueva.

La actualización puede llevar consigo la descarga de nuevos datos del servidor en segundo plano (utilizando XML u otros formatos) o en respuesta a una interacción con el usuario, como hacer clic o mover el cursor.

Al principio las aplicaciones Web utilizaban los *applets (*pequeñas aplicaciones) de Java que se utilizaban para la comunicación remota. A medida que se desarrollaron las tecnologías de los navegadores, otros medios como las IFrames (*inline frame,*marco incorporado) remplazaron a los *applets.*

Más recientemente, XMLHttpRequest se ha añadido a JavaScript, proporcionando una forma para facilitar la transferencia de datos sin necesidad de una nueva petición de página, *applet* o IFrame.

Además de la ventaja de que AJAX utilice elementos que ya son familiares a muchos desarrolladores de aplicaciones Web, AJAX no necesita plugin externos para ejecutarse. Funciona únicamente con la capacidad del navegador para utilizar JavaScript u DHTML, Sin embargo, la dependencia de JavaScript evidencia una de las nuevas desventajas de AJAX: no funciona si el usuario no tiene JavaScript habilitado en su navegador.

Otra desventaja de AJAX es que tiene distintos niveles de admisión de DHTML y JavaScript dependiendo de los navegadores y las plataformas.

Para aquellas aplicaciones en las que los usuarios puedan controlarse (como en las aplicaciones de Internet), AJAX puede programarse para admitir un único navegador en una plataforma determinada.

Sin embargo, cuándo las aplicaciones se abren a usuarios más extensos, como extranet y las aplicaciones de Internet, las aplicaciones AJAX tienen que probarse y normalmente modificarse, para asegurar que se ejecutan de forma idéntica en todos los navegadores y en todos los sistemas operativos.

No es muy probable que AJAX desaparezca a corto plazo, ya que cada día se lanzan al mercado más y más aplicaciones AJAX de alto nivel con enorme éxito, como por ejemplo Google Maps.

Hay que tener en cuenta que AJAX no es en la actualidad un modelo de programación en sí mismo .En realidad se trata de un conjunto de bibliotecas JavaScript. Alguna de estas bibliotecas incluyen componentes reutilizables diseñados para hacer las tareas diarias más fáciles.

2.2.6. FLEX

Flex es un conjunto de librerías para el desarrollo de Interfaces de Usuario que utiliza el lenguaje ActionScript, utilizado para programar aplicaciones Flash. Se explicará con más detalle en el siguiente apartado ya que se trata de la plataforma utilizada para realizar nuestra aplicación

CAPÍTULO 3: PLATAFORMA UTILIZADA

3.1 INTRODUCCIÓN

En marzo de 2002 Macromedia acuñó el término *Rich Internet Applications* (Aplicaciones Ricas de Internet). Por aquél entonces, la idea parecía de algún modo futurista, pero todo ha cambiado y las Aplicaciones Ricas de Internet (RIA) son una realidad ahora mismo y se han convertido en una tecnología básica para la creación de Interfaces ya que nos permite interactuar tanto con un servidor como con los datos y actualizar las Interfaces sin tener que redibujar todo nuevamente.

Macromedia presentó Flex en 2004, lo cuál hizo posible que los desarrolladores escriban aplicaciones para Flash, una plataforma casi omnipresente. Estas aplicaciones pueden beneficiarse de un diseño, funcionalidad y portabilidad mejoradas que cambian de forma espectacular la experiencia del usuario en las aplicaciones Web. Este es un principio básico del concepto Web 2.0, una nueva generación de aplicaciones de Internet basadas en la creatividad y la colaboración.

Desde entonces Macromedia y ahora Adobe, ha sacado al mercado las versiones 1.5, 2 y 3 de Flex.

Con cada nueva versión, la creación de aplicaciones ricas, potentes e intuitivas se ha vuelto más fácil, y se ha aumentado el nivel de las expectativas del usuario sobre las aplicaciones Web.

Pero a pesar del éxito temprano de Flex,; Flex 1 y Flex 1.5 no eran definitivamente un producto del mercado de masas. El precio, la falta de IDE (*Integrated Development Environment*,Entorno Integrado de Desarrollo), las opciones de uso limitadas y otros factores hicieron que las primeras versiones de Flex estuvieran dirigidas específicamente a aplicaciones mayores y más complejas, además de a

desarrolladores y desarrollos más sofisticados. Sin embargo, con las nuevas versiones de productos Flex, todo esto ha cambiado.
Flex 2 salió al mercado en 2006 e hizo del desarrollo de Flex una posibilidad para mucha más gente, ya que incluía un Kit de desarrollo de software (SDK).Con el código abierto de Flex 3 y el anuncio de versiones gratuitas de Flex Builder para estudiantes.

3.2 FLEX

Flex está formado por dos lenguajes: MXML, un lenguaje de marcas basado en XML y ActionScript, el lenguaje de FlashPlayer.
Flex 3 nos ofrece una serie de herramientas y servicios que nos permiten construir y usar las RIA en la plataforma Flash. Flex está formado por varias partes:

1- **ActionScript** 3.0: Un lenguaje de programación orientado hacia el objeto muy potente que avanza en las posibilidades de la plataforma Flash.ActionScript 3.0 está diseñado para crear un lenguaje que en teoría es adecuado para construir las RIA rápidamente. Aunque versiones de ActionScript ofrecían la potencia y la flexibilidad requerida para crear experiencias online atractivas, ActionScript 3.0 avanza aun más en el lenguaje, mejorando la representación y la disposición de desarrollo para facilitar incluso las aplicaciones más complejas con extensos conjuntos de datos y códigos reutilizables completamente orientados hacia el objeto.

2 -**Flash Player 9 (FP 9):** Construido sobre Flash Player , esta generación de Flash Player se centra en la mejora de la ejecución de texto. Para favorecer ésta mejora FP9 incluye una versión de AMV (*ActionScript Virtual*

Machine, Maquina Virtual ActionScript), nueva y totalmente optimizada, conocida como AVM2; AVM2 se construye desde lo más sencillo hasta llegar a trabajar con ActionScript 3.0, la siguiente generación de lenguajes que alimenta Flash Player. La nueva máquina virtual es mucho más rápida y admite informes de errores en el tiempo de ejecución y depuraciones mucho más mejoradas. Flash Player 9 también contendrá AVM1, que ejecuta el código ActionScript 1.0 y 2.0 para ser compatible con versiones anteriores de contenidos ya existentes. A diferencia de las aplicaciones que se construyen utilizando JavaScript, Flash Player es capaz de utilizar un proceso de compilación JIT(*Just In Time,* Justo a tiempo), que hace que se ejecute de forma mas rápida y consuma menos memoria.

3 -**Flex SDK**: Utiliza las bases proporcionadas por FP9 y ActionScript 3.0 y la estructura de una biblioteca de clases extensiva que permite a los desarrolladores utilizar fácilmente las prácticas más adecuadas para construir las RIA con éxito. Flex utiliza un lenguaje basado en XML denominado MXML que permite a los desarrolladores una forma declarativa para manejar los elementos de una aplicación. Los desarrolladores pueden tener acceso a la estructura a través de Flex Builder o la versión gratuita de Flex SDK, que incluye un compilador de líneas de comandos y un programa de depuración, que hace que los desarrolladores puedan utilizar el editor que prefieran y seguir teniendo acceso al compilador o al programa de depuración directamente. En 2007, Adobe anunció el plan de acción de FlexSDK hacia el código abierto.

4 -**Flex Builder 3**: Continuando el éxito de Flex Builder 2, que proporcionaba a los desarrolladores un medio especialmente creado para construir RIA, Flex Builder 3 lleva el IDE (Integrated *Development Environment* Entorno de Desarrollo Integrado) al siguiente nivel.

Construido sobre los estándares más elevados de la industria, el proyecto de código abierto Eclipse, Flex Builder 2 proporciona un excelente medio para el código y la depuración, es una herramienta de diseño útil y rica y promueve las prácticas más adecuadas en el desarrollo del código y de las aplicaciones. Otra ventaja de la plataforma Eclipse es que proporciona un amplio conjunto rico de posibilidades que se pueden ampliar, por lo que pueden realizarse personalizaciones para ampliar el IDE y adecuarse a las necesidades específicas o a las preferencias del desarrollador.

3.3. FLASH

La plataforma Adobe Flash se trata de uno de los tiempos de ejecución más competitivos en el campo de las RIA. La plataforma Flash es actualmente el competidor más importante de AJAX en las RIA.

En su origen fue programado como un plugin para ejecutar animaciones, aunque Flash Player ha evolucionado durante estos años y cada versión añade nuevas características a la vez que todavía se mantienen espacios de utilización muy pequeños. Durante la última década Flash Player se ha convertido casi en omnipresente, con alguna versión instalada en más del 97 por 100 de todos los navegadores Web.

Desde 2002, Macromedia, en la actualidad parte de Adobe, empezó a centrarse en Flash como algo más que una herramienta de animación. Y con la presentación de Flash 6, Macromedia se encontró que con la combinación de la omnipresencia de Flash y la potencia de su lenguaje de texto (ActionScript), de manera que podremos construir aplicaciones basadas totalmente en los navegadores y superar las limitaciones del HTML.

Flash Player nos permite eliminar las incompatibilidades entre plataforma y navegador.

Una de las mejores características de Flash Player es que el contenido y las aplicaciones desarrolladas en cualquier versión de Flash Player se ejecutarán, en la inmensa mayoría de los casos, en cualquier plataforma/navegador que admita la versión de Player.

La mayor desventaja en la construcción de aplicaciones para Flash Placyr era el entorno de creación, que se construía claramente como una herramienta de animación para usuarios que elaboran contenidos interactivos y a la hora de construir RIA para Flash Player existía poca familiaridad con las herramientas.

Esto, unido a la falta de materiales disponibles en 2002 para aprender a utilizar Flash como una plataforma de aplicación, hizo que muchos desarrolladores se mantuvieran al margen de la creación de aplicaciones Flash.Aunque Flash Player todavía es una excelente plataforma para las RIA, la introducción de soluciones como Flex ha simplificado enormemente el proceso de desarrollo y ha reducido el numero de RIAS desarrolladas directamente en Flash Studio.

3.4. ANÁLISIS

A la hora de diseñar la Interfaz nos vimos con la necesidad de escoger la plataforma apropiada para su desarrollo.

La elección de Adobe Flex se debió a que esta plataforma está pensada para el diseño, proporcionando herramientas profesionales que nos permiten crear Interfaces muy cuidadas y completas con gran velocidad además de proporcionarnos una muy buena plataforma de comunicación con servidor. Flex es un marco de trabajo gratuito de código abierto que nos permite crear aplicaciones Web expresivas y muy interactivas que se implantan coherentemente en los

principales exploradores, equipos de sobremesa y sistemas operativos. Ofrece un lenguaje basado en estándares moderno y un modelo de programación que admite los patrones habituales de diseño MXML, un lenguaje declarativo basado en XML que se utiliza para describir el aspecto y comportamiento de la Interfaz de Usuario, y ActionScript, un potente lenguaje de programación orientado a objetos, que se utiliza para crear la lógica de clientes. Asimismo, Flex incorpora una biblioteca muy completa con más de cien componentes de Interfaz de Usuario extensibles y de eficacia demostrada para crear RIA, así como un depurador interactivo de aplicaciones de Flex.

Las aplicaciones de Internet sofisticadas creadas con Flex pueden ejecutarse en el explorador utilizando el software Adobe Flash Player o en el escritorio utilizando Adobe AIR. Esto permite que las aplicaciones de Flex se ejecuten de un modo coherente en todos los exploradores importantes y en múltiples sistemas operativos del escritorio.

CAPÍTULO 4: INTERFAZ

4.1 INTRODUCCIÓN

Cada día se puede observar como la vida del ser humano se ve facilitada por nuevas tecnologías que llegan al hombre de a pie como el mando de la televisión, el horno, el microondas, el teléfono móvil, o Internet. Sin embargo, pocas veces existe la conciencia del proceso de investigación y trabajo que hay detrás de esto. Aunque todo se reduzca a botones y conceptos sencillos, todo esto se debe al desarrollo de lo que hoy en día se conoce como Interfaz. La Real Academia Española de la Lengua define el término "Interfaz" como: "Conexión física y funcional entre dos aparatos o sistemas independientes". En este caso, visto desde el exterior, los dos sistemas independientes que se comunican a través de la Interfaz son el Usuario y el Robot.

En el siguiente apartado se pretende explicar el proceso que se ha seguido para el desarrollo de la Interfaz y la configuración final que esta ha adoptado.

Como ya se ha comentado en el Capítulo 1, el objetivo principal de este proyecto consiste en diseñar una Interfaz de Control de Tareas del Robot HOAP-3.

La Interfaz debe estar orientada a aquellas personas que no están familiarizadas con el Control Automático, la programación de Robots y el entorno Flex de manera que puedan acceder a un ámbito de la tecnología que de otra forma sería más complicado.

Por esta razón se debe diseñar la Interfaz como un entorno amigable en el que, sin necesidad de disponer de unos conocimientos excesivamente técnicos puedan interactuar de manera efectiva.

4.2 DISEÑO DE LA INTERFAZ

A la hora de diseñar nuestra Interfaz se ha buscado que tuviera un aspecto visual sencillo y conceptualmente ajustado a las características de este proyecto.

La apariencia final de la Interfaz es la mostrada en la figura 6 pudiéndose diferenciar un panel de visualización en el cuadrante superior derecho, un panel de control en el cuadrante superior izquierdo, un panel de comandos de alto nivel en el cuadrante inferior izquierdo y un panel de diálogo en el cuadrante inferior derecho. Más adelante explicaremos más detalladamente la funcionalidad de cada uno de ellos.

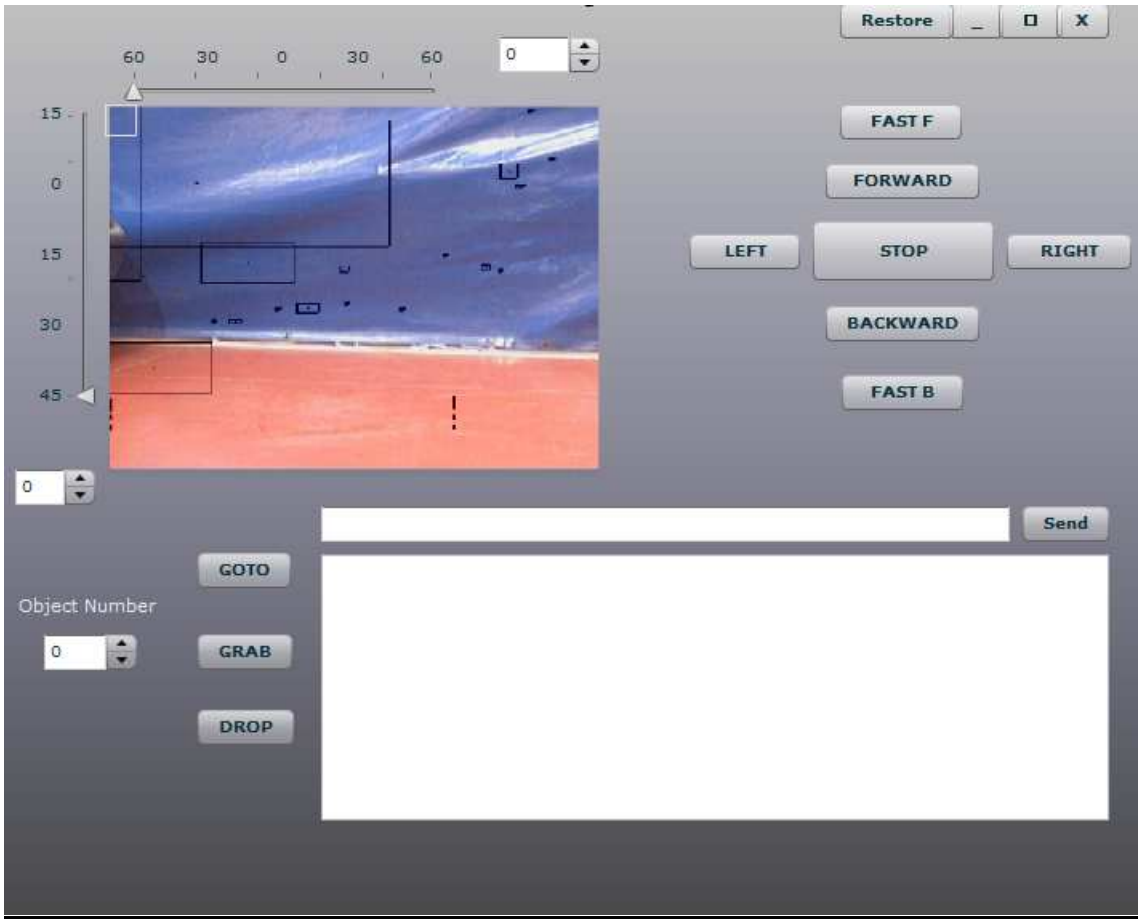

FIGURA 6: Interfaz ROBOT HOAP-3

4.2.1 CÓDIGO DE LA INTERFAZ

En primer lugar creamos un proyecto en Flex Builder como muestra la figura 7. Un proyecto no es otra cosa que una colección de archivos y carpetas que nos ayudaran a organizar el trabajo.

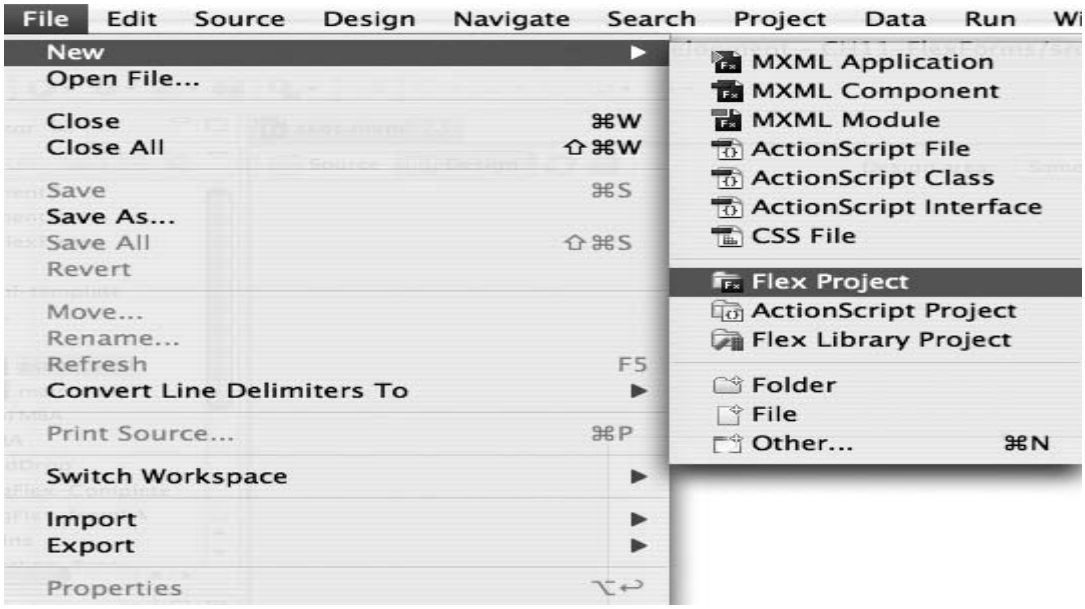

FIGURA 7: Creación proyecto

Una vez creado el proyecto, como se puede distinguir en la figura 8 observamos que tenemos dos opciones a la hora de trabajar con un archivo de aplicación: podemos trabajar en modo *Source* (código) o bien en modo *Design (*Diseño*).*En la mayoría de los casos la vista elegida es una elección personal, pero en algunas ocasiones una funcionalidad determinada sólo estará disponible en una de las vistas en particular.

FIGURA 8: Modos Flex

Hemos marcado la opción *Build Automatically* (Construir Automáticamente) ,como muestra la figura 9 para que Flex compruebe continuamente que se hayan guardado los cambios, los compile antes de guardarlos y los prepare para ejecutarlos; de esta forma los errores de sintaxis se señalan incluso antes de ejecutar la aplicación.

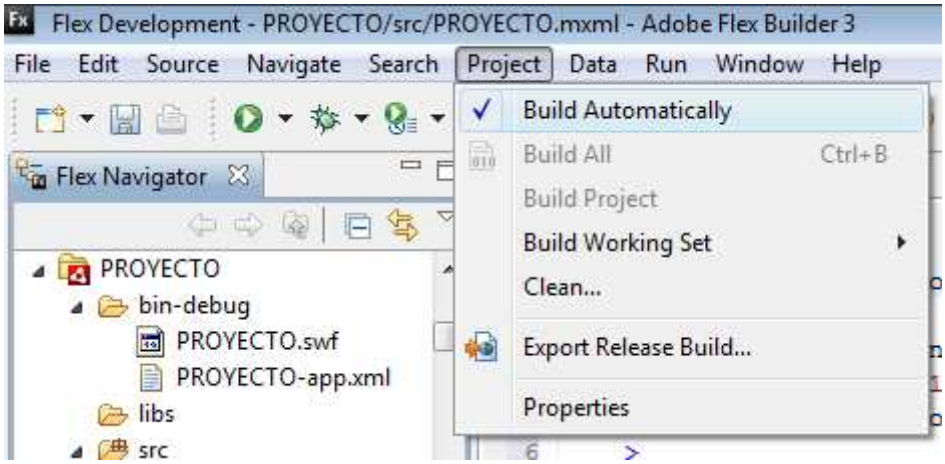

FIGURA 9 : Activación *Build Automatically*

4.2.1.1 CÓDIGO MODO *SOURCE*

El código de nuestra aplicación en modo *source* es el siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute"
        creationComplete="init()"
        borderColor="#010101" backgroundGradientAlphas="[0.0, 0.64]"
        backgroundGradientColors="[#010164, #010101]" width="739" height="586">

                <mx:Script>
                <![CDATA[
                        import mx.controls.Image;
                        import flash.display.Bitmap;
                        import flash.display.BitmapData;
```

```
private const V4L_WIDTH:int = 320;
private const V4L_HEIGHT:int = 240;
private const BUF_SIZE:int = 512;

private function getBitMap(rgbbuf:ByteArray): BitmapData
        {
var c_num:int;
var result:BitmapData = new BitmapData
(V4L_WIDTH,V4L_HEIGHT,false,0x000000);

c_num=0

for (var i:int = 0; i < V4L_HEIGHT; i++)
        {
for (var j:int = 0; j < V4L_WIDTH; j++)
        {
var red:int;
var green:int;
var blue:int;
var rgb:uint;

red = rgbbuf[c_num];
green = rgbbuf[c_num+1];
blue = rgbbuf[c_num+2];

rgb = (red << 16 | green << 8 | blue);

result.setPixel(j, i, rgb);

c_num=c_num+3;

        }
        }
return result;
        }


private function yuv240p_rgb(yuvbuf:ByteArray):ByteArray
{
var rgbbuf:ByteArray = new ByteArray();
var Y:int;
var U:int;
var V:int;
var R:int;
var G:int;
var B:int;
var c_num:int;
var u_ofs:uint;
var v_ofs:uint;
```

```
var y_num:uint;
var u_num:uint;
var v_num:uint;
var uv_cntup_w:uint;
var uv_cntup_h:uint;

u_ofs=V4L_WIDTH*V4L_HEIGHT;
v_ofs=(V4L_WIDTH*V4L_HEIGHT)+((V4L_WIDTH*V4L_HEIGHT)>>2);
y_num=0;
u_num=u_ofs;
v_num=v_ofs;
c_num=0;
uv_cntup_w=0;
uv_cntup_h=0;

for (var j:int=0; j< V4L_HEIGHT; j++)
{

if(uv_cntup_h==0)
{
uv_cntup_h=1;
}
else
{
u_num = u_num - (V4L_WIDTH>>1);
v_num = v_num - (V4L_WIDTH>>1);
uv_cntup_h=0;
}
for (var i:int =0; i < V4L_WIDTH;i++)
{
Y = yuvbuf[y_num];
U = yuvbuf[u_num];
V = yuvbuf[v_num];

R=1.402*(V-128)+Y;
G=-0.344*(U-128)-0.714*(V-128)+Y;
B=1.722*(U-128)+Y;

if(R < 0) R=0;

if(R > 255) R=255;

 if(G < 0) G=0;

 if(G > 255) G=255;

 if(B < 0) B=0;

 if(B > 255) B=255;

rgbbuf[c_num]= R;
rgbbuf[c_num+1]= G;
rgbbuf[c_num+2]= B;
```

```
c_num=c_num+3;
y_num++;

if(uv_cntup_w==0)
{
uv_cntup_w=1;
}
else
{
u_num=u_num+1;
v_num=v_num+1;
uv_cntup_w=0;
}
}
}

return rgbbuf;
}
```

```
private var commandsrv:Socket;
private var videosrv:Socket;
private var t:Timer;
private const TIMER_INTERVAL:int = 300;
private var yuvbuf:ByteArray = new ByteArray;
private var count:int = 0;

private function init():void
{
commandsrv = new Socket()

commandsrv.addEventListener(ProgressEvent.SOCKET_DATA,
socketDatacommand)

commandsrv.addEventListener(IOErrorEvent.IO_ERROR, setErrcommand)

commandsrv.addEventListener(Event.CONNECT, getConncommand)

commandsrv.addEventListener(Event.CLOSE, closeHandlercommand);

commandsrv.addEventListener(SecurityErrorEvent.SECURITY_ERROR,

commandsecurityErrorHandler);

commandsrv.connect("10.59.145.197", 9998)

videosrv = new Socket()

videosrv.addEventListener(ProgressEvent.SOCKET_DATA, socketDatavideo)

videosrv.addEventListener(IOErrorEvent.IO_ERROR, setErrvideo)
videosrv.addEventListener(Event.CONNECT, getConnvideo)
```

```
videosrv.addEventListener(Event.CLOSE, closeHandlervideo);

videosrv.addEventListener(SecurityErrorEvent.SECURITY_ERROR,
videosecurityErrorHandler);

videosrv.connect("10.59.145.197",9999)

t = new Timer(TIMER_INTERVAL);

t.addEventListener(TimerEvent.TIMER, updateTimer);
                                        }

                                private function closeHandlercommand(e:Event):void
                                        {

chatMessage.text += "//****END COMMAND COMUNICATION****\\";

                                        }

                                private function closeHandlervideo(e1:Event):void
                                        {

chatMessage.text += "//****END VIDEO COMUNICATION****\\";

                                        }

                                private function getConncommand(e:Event):void
                                        {

chatMessage.text += "//****START COMMAND COMUNICATION****\\";

                                        }

                                private function getConnvideo(e:Event):void
                                        {

chatMessage.text += "//****START VIDEO COMUNICATION****\\";
t.start();

                                        }

                                private function commandsecurityErrorHandler
(event:SecurityErrorEvent):void
                                        {

 chatMessage.text+= event.text;

                                }

                                private function videosecurityErrorHandler
(event:SecurityErrorEvent):void
                                {
```

```
 chatMessage.text+= event.text;

                                }


                                private function setErrcommand(e:IOErrorEvent):void
                                        {

chatMessage.text += e.text;

                                        }

                                private function setErrvideo (e:IOErrorEvent):void
                                        {

chatMessage.text += e.text;

                                        }

                                private function socketDatacommand (e:ProgressEvent):void
                                        {

trace("socketDataHandler: " + e);
var respuesta:String = commandsrv.readUTFBytes(commandsrv.bytesAvailable);
chatMessage.text += "ROBOT: " + respuesta + "\n";

                                        }
                                private function socketDatavideo (e:ProgressEvent):void
                                        {

var buf:ByteArray = new ByteArray;
var frame_size:Number = V4L_WIDTH*V4L_HEIGHT*3/2;
var div_rest:Number = frame_size % BUF_SIZE;
var div_quot:int = frame_size / BUF_SIZE;
var loop:int;
if (div_rest==0)
{
loop = div_quot;
}
else
{
loop = div_quot + 1;
}
videosrv.readBytes(buf, 0, videosrv.bytesAvailable);
yuvbuf.writeBytes(buf, 0, buf.length);
}

                        private function displayImage():void
                        {
var rgbbuf:ByteArray=yuv240p_rgb(yuvbuf);
var bImageData:BitmapData = new BitmapData(V4L_WIDTH,V4L_HEIGHT,false,0x000000);
bImageData = getBitMap(rgbbuf);
var mBitmap:Bitmap = new Bitmap(bImageData);
```

```
var myImage:Image = new Image();
myImage.source = mBitmap;
canvas.addChild(myImage);
yuvbuf.clear();
rgbbuf.clear();

                    }

                    private function sendMessagecommand(msg:String ):void
                    {
commandsrv.writeUTFBytes(msg);
commandsrv.flush();
chatMessage.text +=msg + "\n"

                    }

                    private function sendMessagevideo(msg1:String ):void
                    {
videosrv.writeUTFBytes(msg1);
videosrv.flush();

                    }

                    private function sendValue(val:int):void
                    {

var Msg:String;
if (val<0)
{
val= Math.abs(val);

Msg='MOVE HEAD DOWN '+val+' DEGREES';
sendMessagecommand(Msg);
}
else
{
Msg='MOVE HEAD UP '+val+' DEGREES';
sendMessagecommand(Msg);
}
}

                    private function sendValue1(val:int):void
                    {

var Msg:String;
if (val<0)
{
val= Math.abs(val);

Msg='MOVE HEAD LEFT '+val+' DEGREES';
sendMessagecommand(Msg);
}
else
```

```
{
Msg='MOVE HEAD RIGHT '+val+' DEGREES';
sendMessagecommand(Msg);
}
}
        private function objectcommand(msg:String):void
                {
var Msg3:String;
Msg3=msg+' OBJECT'+object.value;
sendMessagecommand(Msg3);
                }

                private function updateTimer(evt:TimerEvent):void
                {

sendMessagevideo('getcap0');
displayImage();

                }

                public function minimizeWindow():void
        {

this.stage.nativeWindow.minimize();
        }

       public function maximizeWindow():void
      {

 this.stage.nativeWindow.maximize();

          }

            public function restoreWindow():void

       {

 this.stage.nativeWindow.restore();

        }

       public function closeWindow():void
       {

 this.stage.nativeWindow.close();

        }

                        ]]>

</mx:Script>
<mx:Button x="508" y="136" label="STOP" height="37" width="113" id="emp"
click="sendMessagecommand('STOP')" enabled="true"/>
```

```
<mx:Button x="629" y="144" label="RIGHT" width="78" click="sendMessagecommand('MOVE TURNING RIGHT 15 DEGREES')"/>

<mx:Button x="431" y="144" label="LEFT" click="sendMessagecommand('MOVE TURNING LEFT 15 DEGREES')" width="69"/>

<mx:Button x="516" y="100" label="FORWARD" width="96"  id="ad" click="sendMessagecommand('MOVE WALKING FORWARD 1 STEP')"/>

<mx:Button x="525" y="63" label="FAST F" width="76" click="sendMessagecommand('MOVE WALKING FORWARD 5 STEPS')"/>

<mx:Button x="516" y="189" label="BACKWARD" width="96" click="sendMessagecommand('MOVE WALKING BACKWARD 1 STEP')"/>

<mx:Button x="526" y="232" label="FAST B" width="76" click="sendMessagecommand('MOVE WALKING BACKWARD 5 STEPS')"/>

<mx:Button x="124" y="343" label="GOTO" click="objectcommand('GOTO')"/>

<mx:Button x="124" y="394" label="GRAB" click="objectcommand('GRAB')"/>

<mx:Button x="124" y="441" label="DROP" click="objectcommand('DROP')" />

<mx:TextArea id="chatMessage" editable="false" width="492" height="166"  x="201" y="344"/>

<mx:HSlider x="74" y="25" value="{Hsilder.value}" minimum="-60" maximum="60" tickInterval="25" labels="[60,30,0,30,60]"  snapInterval="1" change="sendValue1(hslider.value)" id="hslider" width="206"/>

<mx:VSlider x="23" y="60" value="{Vsilder.value}" minimum="-45" maximum="15" tickInterval="25" labels="[45,30,15,0,15]" change="sendValue(hslider2.value)" id="hslider2" snapInterval="1" height="194"/>

<mx:NumericStepper x="10" y="291" id="Vsilder" minimum="-50" maximum="50" stepSize="1" click="sendValue(Vsilder.value)" width="50"/>

<mx:NumericStepper x="312" y="21" id="Hsilder" minimum="-50" maximum="50" stepSize="1" click="sendValue1(Hsilder.value)"/>

<mx:TextInput x="201" y="314" id="imput" width="430"/>

<mx:Button x="639" y="314" label="Send" click="sendMessagecommand(imput.text)"/>

<mx:Button label="_" click="minimizeWindow()" x="593" width="34"/>

<mx:Button label="Restore" click="restoreWindow()" x="525"/>

<mx:Button click="maximizeWindow()" x="624" label="□" width="40"/>

<mx:Button label="X" click="closeWindow()" x="660" width="33"/>
```

```
 <mx:Canvas x="66" y="63" width="{V4L_WIDTH}" height="{V4L_HEIGHT}" id="canvas">

</mx:Canvas>

<mx:Button x="604" y="273" label="Button" click="displayImage()"/>

<mx:NumericStepper x="28" y="394" id="object" minimum="0" maximum="10" stepSize="1"/>

<mx:Label x="10" y="367" text="Object Number" fontSize="11" color="#E0EAEC"/>

<mx:Image x="68" y="63" height="232" width="307"
source="file:///C|/Users/ALVARO/Pictures/vision_robot2jpg.jpg"/>


</mx:WindowedApplication>
```

La primera línea de código (<?xml version=”1.0” encoding=”utf-8”?>) es la declaración del tipo de documento XML.MXML es un lenguaje estándar XML,por lo que hay que incluir la declaración del tipo de documento.

La segunda línea del código (<mx:windowedAplication xmlns:mx= ”http://www.adobe.com/2006/mxml” layout=”absolute”>) define la página de una aplicación principal de Flex. La etiqueta <mx:windowedApplication> se refiere al contenedor exterior o portador de todo el contenido de la aplicación Flex, sólo puede haber una única etiqueta <mx:windowedApplication> para cada aplicación Flex.

Dentro de la etiqueta <mx:windowedApplication> el par atributo/valor que parece contener un URL xmlns:mx=”http://www.adobe.com/2006/mxml” define el espacio de nombres para las etiquetas de Flex.Este código indica que el prefijo *mx* está asociado con un conjunto de etiquetas.El valor del atributo que se parece a un URL es al que se denomina URI(*Universal Resource Identifier,*Identificador de Recursos Uniforme) en terminología XML.En un archivo de configuración como *Flex-config.xml,*se lleva cabo una asociación entre este URI y lo que se denomina un archivo de manifiesto, que contiene todas las etiquetas legales que pueden

utilizarse con el prefijo *mx*.Para terminar ,*layout="absolute"* define la forma en la que la aplicación colocará sus nodos descendientes o lo que aparece en la página..
Mediante *backgroundGradientAlphas backgroundGradientColors* definimos el color del fondo de la Interfaz, estas propiedades de forma predeterminada definen un gradiente de fondo gris opaco *backgroundGradientAlphas=[0.1,0.1] y blackgroundGradientColors[0x9CB0BA,0x68808C],* de forma que al cambiar los valores predeterminados Flex calcula el patrón de gradiente entre los dos valores especificados obteniéndose la apariencia mostrada en la figura 6. El evento *creationComplete* se despacha cuando creamos una instancia de un componente y se sitúa correctamente en la aplicación, es el caso de *int()* que aparece más adelante y que será visible en la aplicación cuando se despache *creationComplete* a no ser que la propiedad *visible* esté configurada como *false* que no es el caso.

El bloque <mx:Script> le dice al compilador de Flex que el código en el bloque *Script* es ActionScript. El bloque <![CDATA[]]>dentro del bloque *Script* marca la sección como *carácter data.* Esto le dice al compilador que los datos que aparecen incluidos en el bloque son *characte data,* no datos bien formados en XML, y que no tiene que mostrar errores XML en este bloque. Dentro del bloque <mx:Script> declaramos las variables *commandsrv* (para los comandos) y *videosrv* (para la visualización) como *Socket* que es una clase incorporada de ActionScript que permite abrir una conexión continúa con un servidor .Los datos se envían por la conexión de socket como una cadena y han de estar en formato XML; también declaramos otra variable *t* de la clase *Timer* que nos permite ejecutar código en una secuencia de tiempo especificado además de una constante *TIMER_INTERVAL* de tipo entero con el valor de 300 milisegundos que utilizamos en nuestro temporizador para referenciar la medida de tiempo.

En la función init() de tipo void(no devuelve ningún valor) creamos objetos Socket para *commandsrv* y *videosrv.* Utilizaremos *commandsrv* para el envío de

comandos de control del Robot y *videosrv* para recibir la imagen del Robot en tiempo real.

El método *addEventListener* nos permite "escuchar" un evento y registrar las funciones de controlador de eventos para un objeto especificado tanto para *commandsrv* como para *videosrv* de manera que para el evento DataEvent.DATA llamará a la función traceDatacommand(*command*srv) o trceDatavideo(videosrv), para el evento IOErrorEvent.IO_ERROR llamará a la función setErrcommand(commandsrv) o setErrvideo(videosrv) para el evento Event.CONNECT, llamará a la función getConncommand(commandsrv) o getConnvideo(videosrv) para Event.CLOSE llamará a la función closeHandlercommand(commandsrv) o closeHandlervideo(videosrv) y para TimerEvent.TIMER que crea un evento cada cierto tiempo especificado, llamará a la función updateTimer. Con la sentencia *connect* establecemos una conexión de socket con un puerto de servidor Web. También creamos el objeto *t* de temporizador con la referencia de *TIMER_INTERVAL.*

Las funciones closeHandlercommand y closeHandlervideo de tipo void aceptarán un evento y serán utilizadas para terminar con la comunicación ya que invalida el objeto identificador especificado y muestra en el cuadro de diálogo el mensaje especificado.

Las funciones setErrcommand y setErrvideo de tipo void aceptarán eventos de tipo IOErrorEvent y nos sirve para indicarnos cuando salta un evento en el que se ha producido un fallo en el envío o en la carga de datos, mostrando en el cuadro de comunicación el evento de error en cuestión,

Con las funciones traceDatacommand y traceDatavideo de tipo void se aceptará un evento de tipo DataEvent y hacemos que un detector de eventos se adjunte al objeto Socket que escucha los eventos de tipo *data* y los envía según los va recibiendo a través del cuadro de diálogo

Mediante las funciones getConncommand y getConnvideo establecemos la comunicación y mostramos en el cuadro de comunicaciones el mensaje especificado.

Las funciones sendMessagecommand y sendMessagevideo de tipo void reciben un dato de tipo String y lo envían al servidor además de imprimirlo por la pantalla del cuadro de comunicación de manera que cada vez que llamamos a alguna de estas dos funciones enviaremos el dato tipo *String* que corresponda.

Las funciones sendValuecommand y sendValuevideo de tipo void van a aceptar un valor de tipo entero, se declará una variable de tipo Sting (Msg o Msg1) que contendrá el mensaje a enviar y al valor de tipo entero recibido se le aplicará el valor absoluto; Finalmente se llamará a la función sendMessage(Msg) o sendMessage(Msg1) ,enviándose el mensaje correspondiente.

La función updateTimer de tipo void recibe un evento de tipo *TipeEvent* de manera que cada cierto tiempo especificado por el valor de *TipeEvent* se actualizará la imagen recibida desde el Robot.

Con las funciones públicas de tipo void minimizeWindow, maximizeWindow, restoreWindow y closeWindow obtenemos botones que veremos más adelante que al pulsar nos permitan minimizar, maximizar, restaurar y cerrar la ventana del Interfaz lo que contribuye al dinamismo y a la facilidad de uso de la aplicación.

4.2.1.2 CÓDIGO MODO *DESIGN*

La apariencia de nuestra aplicación en modo *design* es la que se muestra en la figura 10 de manera que podemos apreciar el aspecto final que tendrá nuestra Interfaz.

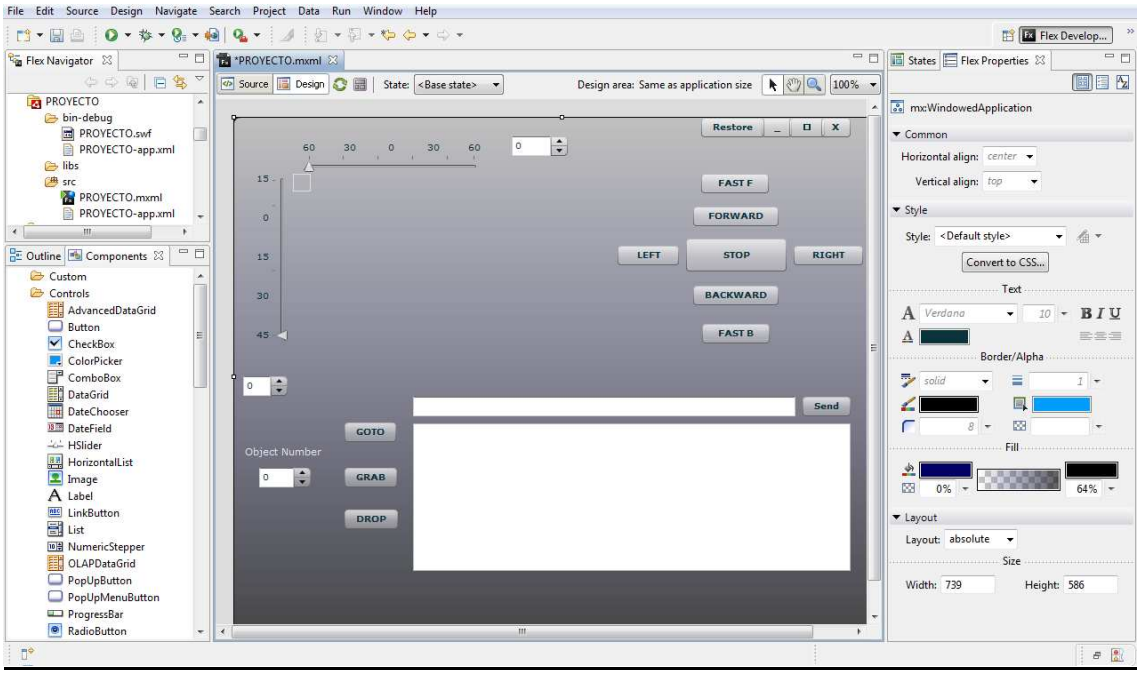

FIGURA 10: Modo *Design*

4.2.1.3 ETIQUETAS

La sintaxis XML correcta nos proporciona dos formas distintas de cerrar una etiqueta. Una es colocar una barra invertida delante del nombre de la etiqueta, y la otra opción es utilizar la barra invertida delante del nombre de la etiqueta que se vuelve a escribir de nuevo.

En nuestro código hemos utilizado normalmente la barra inclinada al final de una etiqueta a no ser que hayamos situado algo dentro del bloque de la etiqueta.

4.2.1.4 EVENTOS

A la hora de entender el método de programación basado en eventos con clases ActionScript personalizadas convendría comentar que Flex utiliza un modelo de programación basado o guiado por eventos. Esto significa que los eventos determinan el flujo de la aplicación. Por ejemplo un usuario que hace clic en el

ratón o los datos que se devuelven desde un servicio Web determinan lo que va a pasar a continuación en la aplicación.

Estos eventos son de dos tipos: los eventos del usuario y los eventos del sistema. Los eventos del usuario son aquellos en los que el usuario hace clic en el botón del ratón o que hace clic en una tecla. Los eventos del sistema incluyen cuando la aplicación es instanciada y se visualiza, además de los componentes que cambian de ser invisibles a visibles. Nosotros somos quienes determinamos lo que tiene que ocurrir cuando se producen determinados eventos escribiendo el código para lo que tiene que suceder.

De esta forma el usuario interactúa con la aplicación, el objeto sobre el que interactúa el usuario despacha un evento, el evento es escuchado y manejado y se ejecuta el código asociado al manejador de eventos correspondiente.

Cuando el usuario hace clic en el botón se despacha el evento *click*. En este caso se ejecuta el ActionScript. Cuando asignamos valores a propiedades proporcionamos dos posibles tipos de valores: escalares o enlaces. Los valores escalares son tipos de datos sencillos como cadenas, números o valores booleanos. Hemos utilizado esto cuando configurábamos los valores *x* e y, el ancho y los valores de etiquetas. También hemos utilizado enlaces para propiedades siempre que hemos utilizado llaves ({}) en un valor.

Cuando se da un valor a un evento, el compilador de Flex entiende que tiene que ser ActionScript. Así que puede escribir ActionScript directamente para el valor del evento sin usar llaves.

4.2.1.5 CONTROLES

Los controles simples son parte de la estructura y ayudan a hacer el desarrollo de las Aplicaciones Ricas de Internet(RIA) más fácil. Utilizar controles nos permite definir de forma más sencilla la apariencia de nuestros botones, el

texto, cuadros y todo lo demás. Flex incluye una biblioteca de clases considerable tanto para los controles simples como para los más complejos. Todas estas clases pueden ser creadas a través de una etiqueta MXML o una clase estándar ActionScript y sus API son accesibles tanto en MXML como en ActionScript. La jerarquía de clases también incluye otras clases que definen el nuevo modelo de evento, además de desplegar atributos que comparten los controles simples.

Situamos los componentes visuales de nuestra aplicación Flex dentro del contenedor *windowedAplicacion* que nos proporciona cuadros delimitadores para el texto, controles, imágenes y otros elementos. Todos los controles simples tienen eventos que pueden utilizarse para responder a las acciones de usuario, como hacer clic en un botón.

4.2.1.6 COMUNICACIÓN SERVIDOR

La comunicación entre Flex y el servidor se realiza mediante sockets en ActionScript de manera que podemos conectarnos a un servidor remoto y nos permite el intercambio de datos sin tener que abrir continuamente nuevas conexiones de servidor.La figura 11 muestra gráficamente la comunicación con el servidor.

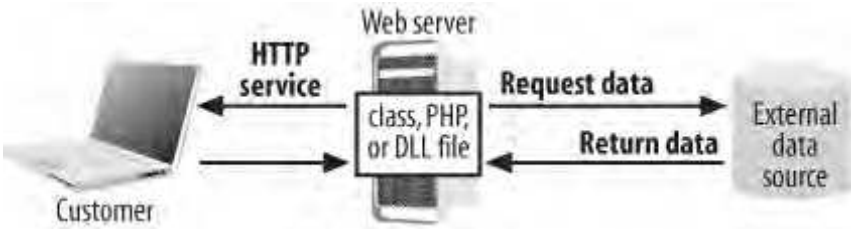

FIGURA 11: Comunicación con servidor

La forma de comunicación con el servidor (servicios Web) utiliza el XML basado en texto para describir como se utilizan los objetos del servidor además del formato en el que se realiza la comunicación

Al hacer llamadas remotas al servidor, estamos llamando a la lógica que reside fuera de Flex con lo que no tenemos control sobre cuando el servidor terminará de ejecutar la petición que hemos realizado. Por tanto, tenemos que utilizar estos eventos y escuchadores de eventos para capturar el momento en el que el servidor ha terminado con la petición. Flex trasmite un evento *result* si el servidor ha sido capaz de procesar la petición con éxito, lo cuál significa que devuelve datos desde la consulta o sencillamente realiza una acción en un servidor. Por el contrario, se transmite un evento *fault* en caso de que se produzca un error durante el proceso de la petición. Tanto el evento *result* como el evento *fault* llevan información adicional, como cualquier dato devuelto o una indicación de porqué se ha producido un fallo. Las peticiones Web se realizan a un servidor externo, lo cuál requiere una cantidad desconocida de tiempo para procesar la petición y devolver los datos apropiados. Por tanto tenemos que "escuchar" los resultados que se devuelven desde el servidor antes de intentar actuar sobre los datos devueltos.

4.2.1.7 TIPOS DE DATOS

Hay dos grupos de datos que habría que diferenciar a la hora de tratar con la memoria Flash Player. Por un lado las primitivas; *int Number, String.*Estos se transmiten por el valor durante llamadas de asignación o de función de forma que se crearían las primitivas y se asignarían sus valores de forma separada. El segundo grupo sería los objetos que transmiten referencias, y en cuyo caso se crearía una única instancia con dos referencias o formas para encontrar el objeto.

4.2.2 PANEL DE VISUALIZACIÓN

Como anteriormente señalamos, en el cuadrante superior izquierdo tendremos un panel como el de la figura, 12 de visualización de la imagen mostrada desde el Robot.

Consta de dos controles deslizantes, dos controles de selección numérica y una pantalla. Con los controles deslizantes fijaremos la posición de la cabeza del Robot siendo el deslizador horizontal el que marca el movimiento de derecha a izquierda y el deslizador vertical por consiguiente el que establecer el movimiento de arriba hacia abajo. El control numérico determina el valor en grados del movimiento de la cabeza.

La imagen mostrará lo que está visualizando el Robot en tiempo real.

Para la ejecución del código hemos utilizado los controles HSlider, VSlider, NummericStepper y el contenedor Canvas.

Usamos los controles Hslider y Vslider para seleccionar un valor moviendo el deslizador entre los extremos de la pista deslizante y los controles NumericStepper para seleccionar un número de un conjunto ordenado configurados de manera que el número seleccionado sea el valor del deslizador correspondiente. Al igual que con el resto de controles, disponen de los atributos *x*, *y* , *height* y *width* que determinan su disposición en el entorno de la Interfaz mediante coordenadas y la asignación de una altura y una anchura .El rango de valores entre los que se mueve nuestro deslizador vienen marcados por los valores *minimun* y *maximun,* el valor correspondiente es el valor númerico escogido en el *NumericStepper* con la sentencia; *value="{Hslider.value}"* para el deslizador horizontal y *value=* "*{VSlider.value}"* para el deslizador vertical; ,de esta forma en el momento en el que hacemos clic en el control *NumericStepper* el valor numérico del mismo pasa a ser el valor del deslizador. Cada componente tiene su nombre de instancia asignado (*id).*En el momento en el que un nuevo valor es introducido en el deslizador

llamamos en el caso del deslizador horizontal a la función *sendValue(hslider.value)* enviándole el entero correspondiente al valor del deslizador vertical y en el caso del deslizador horizontal a la función *sendValue(hslider2.value)* enviándole el valor del entero correspondiente a la posición del deslizador horizontal. Las funciones *sendValue()* y *sendValue1()* se encargarán de comprobar si el número recibido en la función es positivo o negativo enviando el mensaje correspondiente como ya hemos visto. Usamos el control de deslizamiento horizontal para controlar el movimiento de la cabeza del Robot de derecha a izquierda siendo los números positivos del deslizador el número de grados a deslizar hacia la derecha y los números negativos el número de grados a girar hacia la izquierda. En el caso del deslizador vertical, el control será del movimiento de la cabeza del Robot hacia arriba y hacia abajo, siendo los números positivos del deslizador el número de grados que ha de deslizarse hacia arriba y los números negativos el número de grados que ha de deslizarse hacia abajo. Las funciones *sendValue* y *sendValue1* se encargaran de enviar la información correcta en cada uno de los casos.

```
<mx:HSlider x="74" y="25" value="{Hsilder.value}" minimum="-50" maximum="50"
tickInterval="25" labels="[50,25,0,25,50]"  snapInterval="1" change="sendValue1(hslider.value)"
id="hslider" width="206"/>

<mx:VSlider x="23" y="60" value="{Vsilder.value}" minimum="-50" maximum="50"
tickInterval="25" labels="[50,25,0,25,50]" change="sendValue(hslider2.value)" id="hslider2"
snapInterval="1" height="194"/>

<mx:NumericStepper x="10" y="291" id="Vsilder" minimum="-50" maximum="50" stepSize="1"
click="sendValue(Vsilder.value)"/>

<mx:NumericStepper x="312" y="21" id="Hsilder" minimum="-50" maximum="50" stepSize="1"
click="sendValue1(Hsilder.value)"/>
```

Para llevar la imagen del Robot a nuestra Interfaz hemos creado un *bitmapdata* que nos permite trabajar con los datos (pixeles) de un objeto de mapa de bits ya que contiene una matriz de datos de pixeles.Se han utilizado los métodos de la clase *bitmapdata* para crear el tamaño de la imagen de mapa de bits acorde a la anchura y altura de la imagen del Robot.Para cambiar la apariencia de la imagen de mapa de bits en un nivel de pixel, obtenemos los valores de color de los pixeles contenidos en nuestra área. Utilizamos la función *getBitMap* para leer los valores de los pixeles.

A partir de aquí ajustamos la información del valor del color RGB de los pixeles recibida del Robot y mediante la función displayImage() la representamos en nuestra Interfaz .

El temporizador lo vamos a usar para que cada 0.3 segundos se vuelvan a recibir datos por parte del Robot y se muestre la imagen nuevamente.De esta forma estaríamos viendo la imagen de la cámara del Robot .

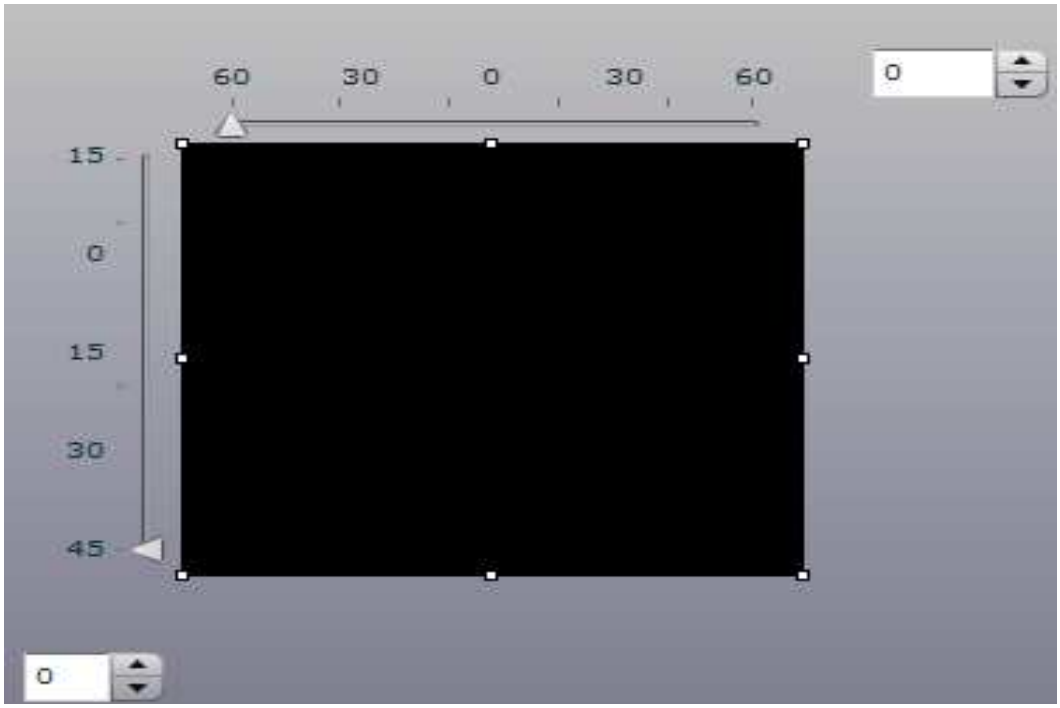

FIGURA 12: Panel Visualización

4.2.3. PANEL DE CONTROL

A través del panel de control mostrado en la figura 13 situado en el cuadrante superior derecho daremos las ordenes pertinentes al Robot de forma que avance un paso hacia delante, hacia atrás, gire bien a la derecha bien a la izquierda e incluso que avance 5 pasaos hacia delante o hacia atrás. También dispondremos de la opción de ordenarle que pare.

Para la implementación de éste panel hemos hecho uso de los controles *Button:*

El control *Button* es un botón rectangular que puede tener una etiqueta de texto, un icono o ambas en su cara. Los botones utilizan detectores de eventos para realizar una acción cuando seleccionamos el control, al hacer clic con el ratón en un control *Button* habilitado se distribuye un evento *click.* Podemos personalizar el aspecto de un *Button* y cambiar la funcionalidad de cada uno de ellos independientemente. De esta forma los atributos *x* e *y* determinan las coordenadas en las que se sitúa el botón *height* y *width* determinan la altura y la anchura ,*id* es el nombre de instancia asignado y en el atributo *click* introducimos el nombre de la función de ActionScript que queremos ejecutar cuando este botón sea pulsado de manera que cada uno de ellos ocupará una posición concreta y tendrá unas dimensiones especificas.

En nuestra Interfaz, en el momento que son pulsados llamarán a la función *sendMessage* que se encargará de enviar la información específica correspondiente a cada botón.

```
<mx:Button x="483" y="136" label="STOP" height="37" width="113"        id="emp"
click="sendMessage('STOP')" enabled="true"/>

<mx:Button x="615" y="144" label="RIGHT" width="78" click="sendMessage('MOVE TURNING
RIGHT 15 DEGREES')"/>

<mx:Button x="387" y="144" label="LEFT" click="sendMessage('MOVE TURNING LEFT 15
DEGREES')" width="69"/>
```

```
<mx:Button x="491" y="94" label="FORWARD" width="96"   id="ad" click="sendMessage('MOVE WALKING FORWARD 1 STEP')"/>

<mx:Button x="501" y="51" label="FAST F" width="76" click="sendMessage('MOVE WALKING FORWARD 5 STEPS')"/>

<mx:Button x="491" y="189" label="BACKWARD" width="96" click="sendMessage('MOVE WALKING BACKWARD 1 STEP')"/>

<mx:Button x="501" y="232" label="FAST B" width="76" click="sendMessage('MOVE WALKING BACKWARD 5 STEPS')"/>
```

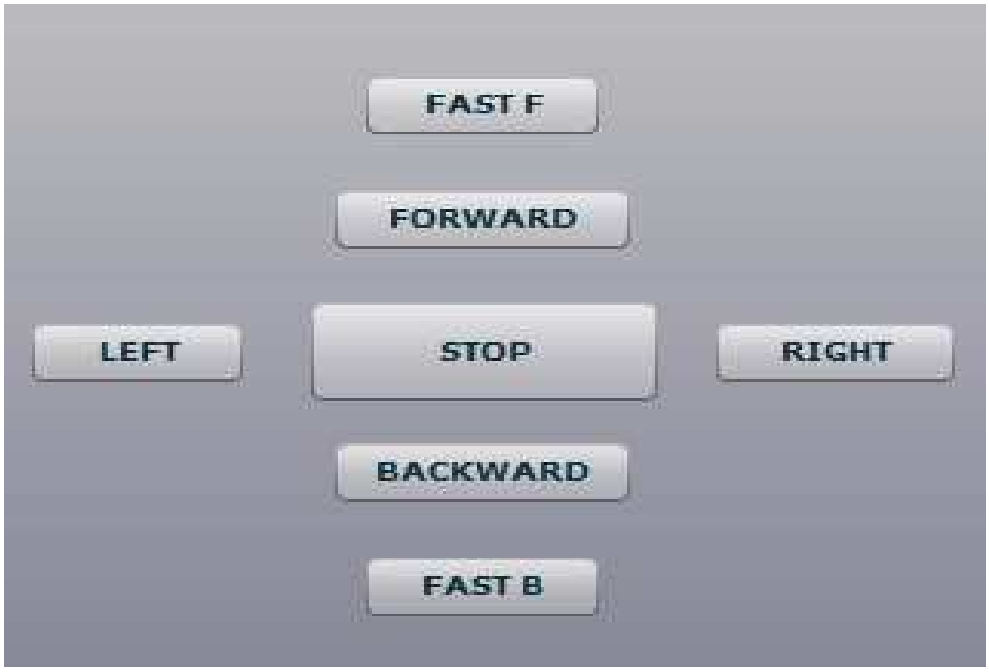

FIGURA 13: Panel de Control

4.2.4. PANEL DE DIÁLOGO

Con el panel de diálogo mostrado en la figura 14 establecemos la comunicación efectiva con el Robot de forma que podamos observar los comandos que se están enviando y las respuestas que vamos recibiendo en el área de texto inferior y que además podamos enviar indicando en el área de texto superior y tras pulsar el botón , los comandos que deseemos.

A la hora de ejecutar el código vamos a utilizar los controles *TextArea, TextImput* y *Button.*

Los controles *TexImput* son líneas de texto de campo único editable que nosotros utilizaremos para enviar información. En primer lugar lo que hacemos es asignar los valores a los atributos *x* e *y* de forma que quede situado en la Interfaz y le asignaremos una *id* ya que nuestra intención es referenciar mediante un botón el *TextImput* correspondiente .Para cada uno de los canales de envío y recepción de información dispondremos de esta opción para poder mandar la información que nosotros creamos oportuna. El botón lo editamos de forma que queden posicionados con los atributos *x* e *y* con el texto *Send* y creando el evento de llamada a la función *sendMessage* de manera que al hacer clic en el botón correspondiente se envíe la información escrita en la línea de texto de los *TextImput* y se muestren por el cuadro de comunicación.

El control TextArea es un campo de texto de varias líneas con borde y barra de deslizamiento en el que nosotros veremos la información que intercambiamos en la comunicación .Al igual que ocurría en el caso de los botones disponemos de los atributos *x ,y, height, width* que determinan su posición altura y anchura dentro de la Interfaz, una *id* con el nombre de instancia asignado para poder hacer referencia en otra parte del código y por último el atributo editable que indica si el usuario

tiene permiso para editar el texto de este control ,en este caso, especificado como *false.*

```
<mx:TextInput x="174" y="314" id="imput"/>
<mx:Button x="342"y="314"label="send"click="sendMessage(imput.text)"/>
<mx:TextArea id="chatMessage" editable="false" width="256" height="166"  x="173" y="344"/>
```

FIGURA 14: Panel de Diálogo

4.2.5. PANEL DE COMANDOS DE ALTO NIVEL

Utilizaremos el panel de comandos de alto nivel como el que aparece en la figura 15 para indicar acciones como agarrar, soltar o ir hacia algún objeto.

Al igual que con el panel de control para la implementación de código vamos a hacer uso de los controles *Button* de forma que al hacer clic sobre cada uno de ellos referencien a la función *sendMessage* y manden al Robot la orden correspondiente; "ir hacia un objeto","agarrar un objeto" o "soltar un objeto" de

manera que éste las ejecute. Mediante el control *NumericStepper* vamos a poder seleccionar un número de forma que le indiquemos al Robot sobre que objeto debe realizar la acción. Esta asociación de objeto y acción la efectuamos mediante la función *objectcommad()*.

Por último se ha utilizado un control *label* que muestra una línea de código no editable.

```
<mx:Button x="124" y="343"label="GOTO" click="objectcommand('GOTO')"/>
<mx:Button x="124" y="394"label="GRAB" click="objectcommand('GRAB')"/>
<mx:Button x="124" y="441"label="DROP"click="objectcommand('DROP')" />
<mx:NumericStepper x="28" y="394" id="object" minimum="0" maximum="10"
stepSize="1"/>

<mx:Label x="10" y="367" text="Object Number" fontSize="11"
color="#E0EAEC"/>
```

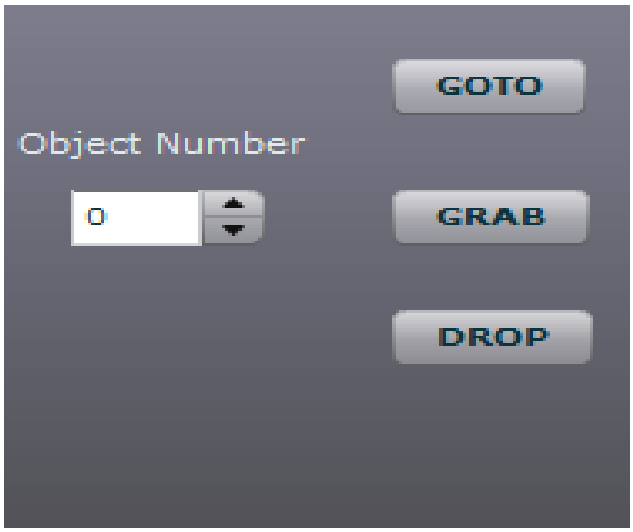

FIGURA 15: Panel Comandos Alto Nivel

CAPÍTULO 5: IMPLEMENTACIÓN EN EL HOAP

5.1 ROBOT HOAP

Fujitsu Automation en colaboración con Fujitsu Lab han desarrollado el Robot Humanoide en miniatura HOAP (Humanoid for Open Architecture Platform).
La primera versión de este Robot HOAP-1, que salió a mercado en el año 2001, seguida por la segunda versión HOAP-2 en el año 2003, han dado lugar al desarrollo de la tercera versión de dicho Robot, HOAP-3, en el año 2005.La figura 16 muestra la imagen del Robot HOAP-3

FIGURA 16: Robot HOAP-3

Desde la venta de la primera versión HOAP-1 en 2001, la serie de Robots Humanoides HOAP ha sido cada vez más utilizada en los departamentos de ingeniería mecánica de las universidades, así como en las áreas de investigación Robótica.

El Robot HOAP-3 posee 28 grados de libertad (GDL) distribuidos como se muestra en la figura 17. Tiene una altura de 60 cm y un peso aproximado de 8 Kg.

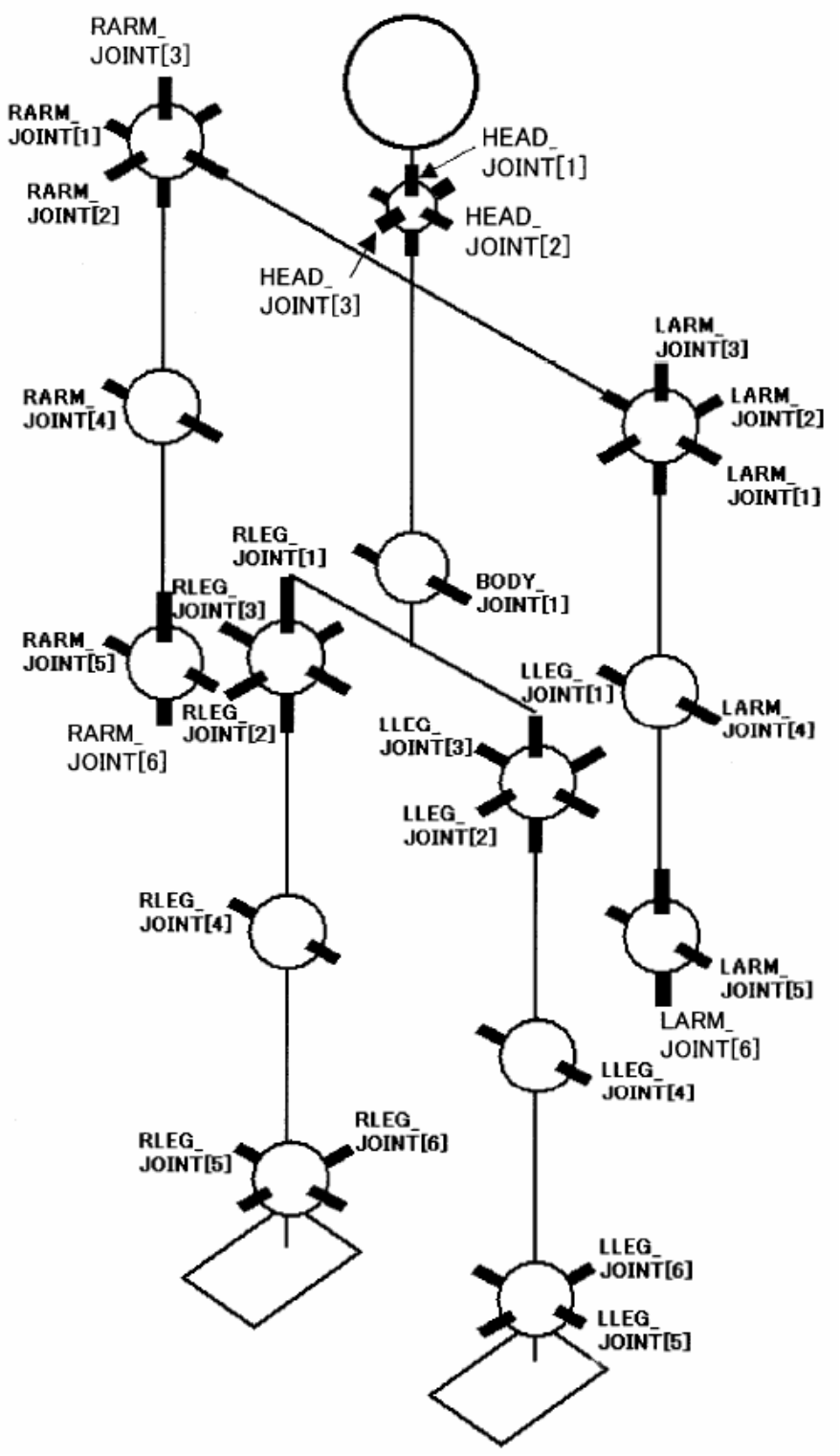

FIGURA 17: Distribución Grados de Libertad

6 GDL／Pierna × 2

5 GDL／ Brazo × 2

1 GDL／Cintura × 1

1 GDL／Mano × 2

3 GDL／Cuello × 1

_____

Total: 28 GDL

El Robot HOAP-3 tiene varios sensores, entre otros de aceleración, de fuerza, o para medición de distancias a través de infrarrojos .

Está dotado de una cámara, un micrófono, un altavoz y dos LED en la cabeza. La versión 3 además incluye características tales como la de mostrar emociones mediante un LED, reconocer imágenes y sonidos, sintetizar la voz y la posibilidad de comunicarse con un humano en un intento de de diálogo natural fluido.

La información de Interfaz de hardware y software es de código abierto, por lo que puede ser programado libremente.

Todo ello funciona sobre RTLinux y se proporcionan toda clase de herramientas para este sistema operativo con el fin de poder programar el Robot y añadir funcionalidades adicionales dando control total al usuario.

5.2. COMUNICACIÓN CON EL HOAP

Para explicar la captura de la imagen del Robot he recurrido al Sistema de Visión del Humanoide HOAP-3 para la Detención e Identificación de Objetos Mediante Librerías OpenCV descrito por A. PEÑA, D. HERNÁNDEZ GARCÍA, M. GONZÁLEZ-FIERRO, P. PIERRO y C. BALAGUER.

El sistema de visión del Robot Humanoide se implementó con una arquitectura sencilla de cliente servidor. En la figura 18 se muestra la arquitectura del sistema implementado.

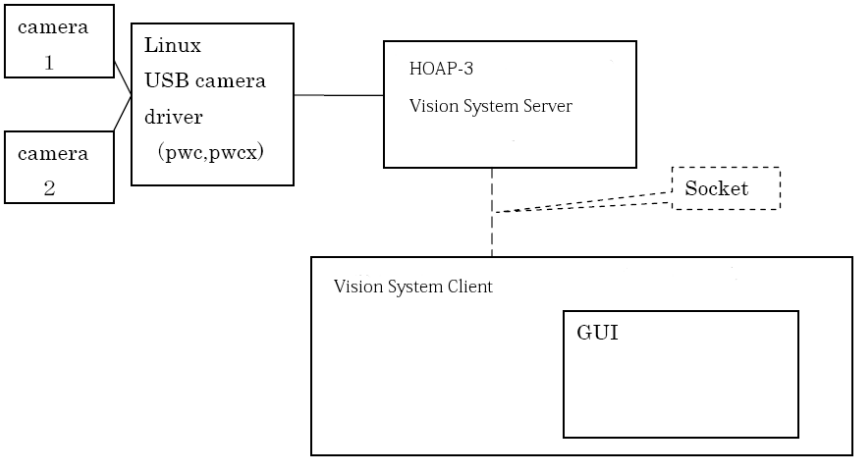

FIGURA 18: Arquitectura Sistema

El robot HOAP-3 cuenta con un sistema de dos cámaras que nos permite conseguir una visión estero. El servidor de visión, PC interno del HOAP-3, se encarga de capturar y enviar la captura de imagen de las cámaras como un array de char. El cliente recibe la información de la imagen del Robot en el formato YUV240p y luego convierte este formato a la estructura IplImage requerida por algoritmos de detección e identificación.

5.2. CONTOL DEL ROBOT MEDIANTE LA INTERFAZ

En este apartado hemos evaluado el funcionamiento de nuestra Interfaz desarrollando algunos comandos de control.

Por un lado se muestra la apariencia de ejecución de la Interfaz y por el otro se observa la ejecución del comando en el Robot.

En primer lugar le hemos indicado que avance un paso hacia delante, pulsando en nuestra Interfaz el botón *FORWARD.*

Podemos observar en la figura 19 como en el cuadro de comunicación se muestra el comando enviado: *MOVE WALKING FORWARD 1 STEP,* y la respuesta recibida del Robot: *OK COMMAND WLK COMPLETED.*

También vemos en la figura 20 al Robot grabado desde una posición frontal como ejecuta la acción de andar un paso hacia delante.

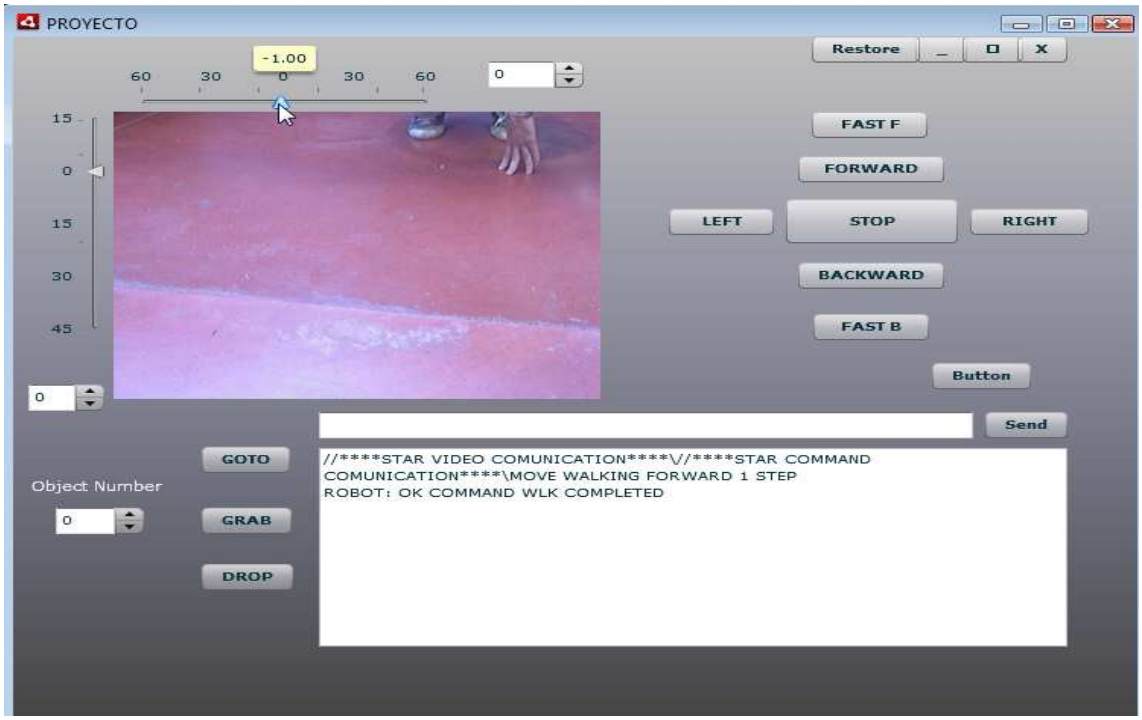

FIGURA 19: Interfaz para Paso

FIGURA 20: HOAP dando paso

En segundo lugar vamos a indicarle al Robot que gire la cabeza cinco grados a la derecha y seis hacia la izquierda, de manera que podemos apreciar en la figura 21 el comando enviado: *MOVE HEAD LEFT 5 DEGREES* y *MOVE HEAD UP 6 DEGREES* y la correspondiente respuesta del Robot: *OK COMMAND MHL COMPLETED* y *OK COMMAND MHU COMPLETED.*

En este caso vemos la visión desde nuestra Interfaz (figura 21y figura 23) y la posición del Robot antes y después de ejecutar la acción mostrados en la figura 22 y 24.

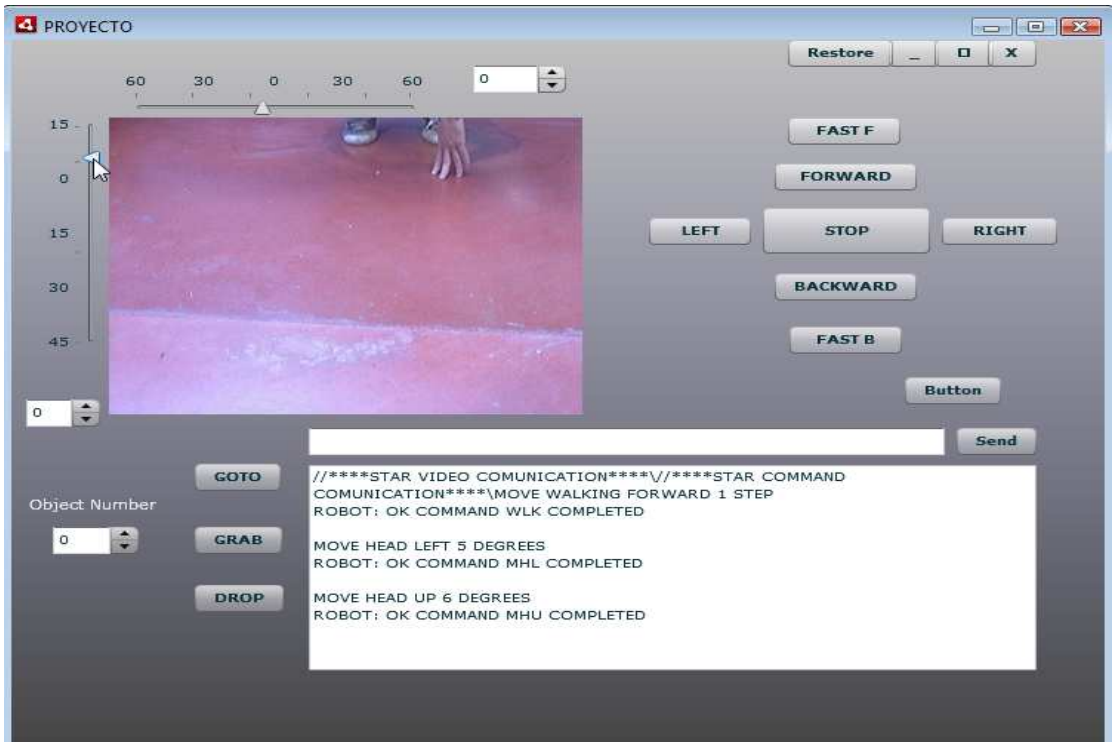

FIGURA 21: Interfaz para movimiento de cabeza

FIGURA 22: HOAP antes giro cabeza

FIGURA 23: Interfaz ejecución movimiento cabeza

FIGURA 24: HOAP tras giro cabeza

Por último le indicamos que haga un movimiento de giro hacia la derecha de 15 grados ,observándose en el cuadro de diálogo de la figura 25 el comando enviado: *MOVE TURNING RIGHT 15 DEGREES.* En la figura 26 observamos al Robot una vez girado.

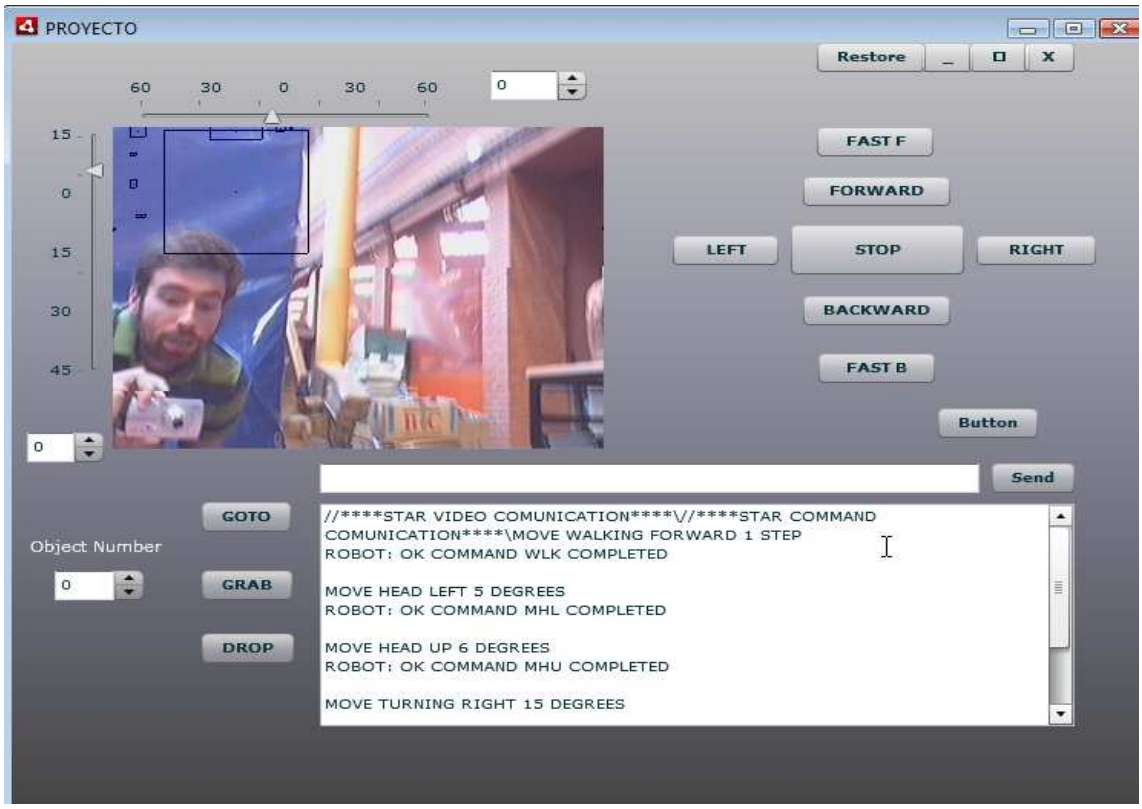

FIGURA 25: Interfaz giro a la derecha

FIGURA 26: HOAP girado a la derecha

CAPÍTULO 6: CONCLUSIONES

En primer lugar hay que decir que se ha cumplido el principal objetivo de este proyecto que consistía en le creación de una Interfaz de control de movimientos del Robot HOAP-3.Dicha Interfaz permitiría la comunicación bidireccional con el dispositivo del Robot de manera que éste recibiera las ordenes determinadas en la Interfaz y procediera a su ejecución enviando un mensaje de conformidad o de error en el caso de que se produjera.

Hemos conseguido diseñar una Interfaz práctica y funcional y además hemos conseguido una comunicación fluida con el Robot.

La Interfaz Gráfica se ha desarrollado con la herramienta Flex. Flex es una plataforma software orientada a objetos para la creación de Interfaces Graficas de Usuario dotada de una muy buena herramienta de comunicación con el servidor.

Por último han sido comprobadas las aplicaciones desarrolladas en nuestro Interfaz dando un funcionamiento correcto en el Robot.

A título personal la elaboración de este proyecto ha sido una muy buena experiencia ya que me ha permitido involucrarme en la Robótica Humanoide, un campo bastante puntero y en el que intervienen la mayor parte de las ramas de la ingeniería: electrónica, cinemática, dinámica, regulación y control….también he tenido la oportunidad de ver el desarrollo de un gran proyecto de investigación como es el del Robot Humanoide HOAP-3 llevado a cabo por el departamento de Sistemas y Automática de la Universidad Carlos III de Madrid.

CAPÍTULO 7: BIBLIOGRAFÍA

Lorenzo Blasi:Deliable 3.8@M36:Final Report on software concepts for ROBOT@CWE

Barrientos A.; Peñín L.F., Balaguer C., Aracil R.; *Fundamentos de Robótica*; McGraw-Hill, 1997.

Jeff Tapper,Michael Labriola, Mathew Boles, James Talbot; *Adobe Flex 3* ;Anaya,2007

"HOAP-3 Instruction Manual". FUJITSU.

MANUALES:

Manual de uso de la herramienta Flex: *Getting Started with Flex*. Jack Herrington and Emily Kim. Adobe Developer Library

Manual de uso de la herramienta Flex: *Programmingas3*. Adobe Developer Library

Manual de uso de la herramienta Flex: *Beginers Guide Flex 3*. McGraw Hill

Manual de uso de la herramienta Flex: *Getting Started with Flex*. Jack Herrington and Emily Kim. Adobe Developer Library

Manual de uso de la herramienta Flex: *Dev Guide Flex 3*. Adobe Developer Library

"HOAP-3 Instruction Manual". FUJITSU.

REFERENCIAS WEB:

Página Web de Microbótica: www.microbotica.com (Septiembre 2010)

Página Web de Acroname: www.acroname.com (Agosto 2010)

Página Web de Dr. Robot: www.drrobot.com (Septiembre 2010)

Página Web de Dpac Technology: www.dpactech.com (Septiembre 2010)

Pagina Web de Adobe : www.adobe.com/es/products/flex/ (Junio 2010)

Pagina Web :www.monografias.com (Septiembre 2010)

Pagina Web de Wikipedia: http://es.wikipedia.org/wiki/Adobe_Flex
http://es.wikipedia.org/wiki/Rob%C3%B3tica (Mayo 2010)

Pagina Web RAE: www.rae.es (Mayo 2010)

ANEXO 1

CÓDIGO

```
<?xml version="1.0" encoding="utf-8"?>
<mx:WindowedApplication xmlns:mx="http://www.adobe.com/2006/mxml"
layout="absolute"
      creationComplete="init()"
      borderColor="#010101" backgroundGradientAlphas="[0.0, 0.64]"
      backgroundGradientColors="[#010164, #010101]" width="739"
height="586">

            <mx:Script>
            <![CDATA[
                  import mx.controls.Image;
                import flash.display.Bitmap;
                  import flash.display.BitmapData;



                        private const V4L_WIDTH:int = 320;
                  private const V4L_HEIGHT:int = 240;
                  private const BUF_SIZE:int = 512;

                        private function
getBitMap(rgbbuf:ByteArray):BitmapData
                        {
                        var c_num:int;
                        var result:BitmapData = new
BitmapData(V4L_WIDTH,V4L_HEIGHT,false,0x000000);

                        c_num=0

                        for (var i:int = 0; i < V4L_HEIGHT; i++)
                        {
                        for (var j:int = 0; j < V4L_WIDTH; j++)
                        {
                        var red:int;
                        var green:int;
                        var blue:int;
                        var rgb:uint;

                        red = rgbbuf[c_num];
                        green = rgbbuf[c_num+1];
                        blue = rgbbuf[c_num+2];

                        rgb = (red << 16 | green << 8 | blue);

                        result.setPixel(j, i, rgb);

                        c_num=c_num+3;

                        }
                        }
                        return result;
                        }
```

```
                private function
yuv240p_rgb(yuvbuf:ByteArray):ByteArray
                {
                var rgbbuf:ByteArray = new ByteArray();
                var Y:int;
                var U:int;
                var V:int;
                var R:int;
                var G:int;
                var B:int;
                var c_num:int;
                var u_ofs:uint;
                var v_ofs:uint;
                var y_num:uint;
                var u_num:uint;
                var v_num:uint;
                var uv_cntup_w:uint;
                var uv_cntup_h:uint;

                u_ofs=V4L_WIDTH*V4L_HEIGHT;

v_ofs=(V4L_WIDTH*V4L_HEIGHT)+((V4L_WIDTH*V4L_HEIGHT)>>2);
                y_num=0;
                u_num=u_ofs;
                v_num=v_ofs;
                c_num=0;
                uv_cntup_w=0;
                uv_cntup_h=0;

                for (var j:int=0; j< V4L_HEIGHT; j++)
                {

                if(uv_cntup_h==0)
                {
                uv_cntup_h=1;
                }
                else
                {
                u_num = u_num - (V4L_WIDTH>>1);
                v_num = v_num - (V4L_WIDTH>>1);
                uv_cntup_h=0;
                }
                for (var i:int =0; i < V4L_WIDTH;i++)
                {
                Y = yuvbuf[y_num];
                U = yuvbuf[u_num];
                V = yuvbuf[v_num];

                R=1.402*(V-128)+Y;
                G=-0.344*(U-128)-0.714*(V-128)+Y;
                B=1.722*(U-128)+Y;
```

```
if(R < 0) R=0;
if(R > 255) R=255;
if(G < 0) G=0;
if(G > 255) G=255;
if(B < 0) B=0;
if(B > 255) B=255;

rgbbuf[c_num]= R;
rgbbuf[c_num+1]= G;
rgbbuf[c_num+2]= B;
c_num=c_num+3;
y_num++;

if(uv_cntup_w==0)
{
uv_cntup_w=1;
}
else
{
u_num=u_num+1;
v_num=v_num+1;
uv_cntup_w=0;
}
}
}

return rgbbuf;
}
```

```
        private var commandsrv:Socket;
        private var videosrv:Socket;
        private var t:Timer;
        private const TIMER_INTERVAL:int = 300;
        private var yuvbuf:ByteArray = new ByteArray;
        private var count:int = 0;

        private function init():void
        {
      commandsrv = new Socket()

      commandsrv.addEventListener(ProgressEvent.SOCKET_DATA,
socketDatacommand)

      commandsrv.addEventListener(IOErrorEvent.IO_ERROR,
setErrcommand)
                commandsrv.addEventListener(Event.CONNECT,
getConncommand)
                commandsrv.addEventListener(Event.CLOSE,
closeHandlercommand);

      commandsrv.addEventListener(SecurityErrorEvent.SECURITY_ERROR,
commandsecurityErrorHandler);
                commandsrv.connect("10.59.145.197", 9998)
                videosrv = new Socket()
```

```
      videosrv.addEventListener(ProgressEvent.SOCKET_DATA,
socketDatavideo)

      videosrv.addEventListener(IOErrorEvent.IO_ERROR, setErrvideo)
                        videosrv.addEventListener(Event.CONNECT,
getConnvideo)
                        videosrv.addEventListener(Event.CLOSE,
closeHandlervideo);

      videosrv.addEventListener(SecurityErrorEvent.SECURITY_ERROR,
videosecurityErrorHandler);
                        videosrv.connect("10.59.145.197",9999)
                        t = new Timer(TIMER_INTERVAL);
                t.addEventListener(TimerEvent.TIMER, updateTimer);
                        }

                        private function
closeHandlercommand(e:Event):void
                        {

                        chatMessage.text += "//****END COMMAND
COMUNICATION****\\";

                        }

                        private function
closeHandlervideo(e1:Event):void
                        {

                        chatMessage.text += "//****END VIDEO
COMUNICATION****\\";

                        }

                        private function getConncommand(e:Event):void
                        {

                        chatMessage.text += "//****START COMMAND
COMUNICATION****\\";

                        }

                        private function getConnvideo(e:Event):void
                        {

                        chatMessage.text += "//****START VIDEO
COMUNICATION****\\";
                        t.start();

                        }

                        private function
commandsecurityErrorHandler(event:SecurityErrorEvent):void
                        {
```

```
                    chatMessage.text+= event.text;

                    }

                    private function
videosecurityErrorHandler(event:SecurityErrorEvent):void
                    {

                    chatMessage.text+= event.text;

                    }


                          private function
setErrcommand(e:IOErrorEvent):void
                          {

                          chatMessage.text += e.text;

                          }

                          private function
setErrvideo(e:IOErrorEvent):void
                          {

                          chatMessage.text += e.text;

                          }

                          private function
socketDatacommand(e:ProgressEvent):void
                          {

                          trace("socketDataHandler: " + e);
                          var respuesta:String =
commandsrv.readUTFBytes(commandsrv.bytesAvailable);
                          chatMessage.text += "ROBOT: " + respuesta +
"\n";


                          }
                          private function
socketDatavideo(e:ProgressEvent):void
                          {

                          var buf:ByteArray = new ByteArray;
                          var frame_size:Number =
V4L_WIDTH*V4L_HEIGHT*3/2;
                          var div_rest:Number = frame_size % BUF_SIZE;
                          var div_quot:int = frame_size / BUF_SIZE;
                          var loop:int;

                          if (div_rest==0)
                          {
                                loop = div_quot;
```

```
                        }
                        else
                        {
                              loop = div_quot + 1;
                        }
                        videosrv.readBytes(buf, 0,
videosrv.bytesAvailable);
                        yuvbuf.writeBytes(buf, 0, buf.length);
                        }

                        private function displayImage():void
                        {
                        var rgbbuf:ByteArray=yuv240p_rgb(yuvbuf);
                        var bImageData:BitmapData = new
BitmapData(V4L_WIDTH,V4L_HEIGHT,false,0x000000);
                        bImageData = getBitMap(rgbbuf);
                        var mBitmap:Bitmap = new Bitmap(bImageData);
                        var myImage:Image = new Image();
                        myImage.source = mBitmap;
                        canvas.addChild(myImage);
                        yuvbuf.clear();
                        rgbbuf.clear();

                        }

                        private function sendMessagecommand(msg:String
):void
                        {

                        commandsrv.writeUTFBytes(msg);
                        commandsrv.flush();
                        chatMessage.text +=msg + "\n"

                        }

                        private function sendMessagevideo(msg1:String
):void
                        {
                        videosrv.writeUTFBytes(msg1);
                        videosrv.flush();

                        }

                        private function sendValue(val:int):void
                        {

                        var Msg:String;
                        if (val<0){
                        val= Math.abs(val);

                        Msg='MOVE HEAD DOWN '+val+' DEGREES';
                        sendMessagecommand(Msg);
                    }
                    else
                    {
```

```
                    Msg='MOVE HEAD UP '+val+' DEGREES';
                    sendMessagecommand(Msg);
                       }
                       }

                       private function sendValue1(val:int):void
                       {

                       var Msg:String;
                       if (val<0){
                       val= Math.abs(val);

                       Msg='MOVE HEAD LEFT '+val+' DEGREES';
                       sendMessagecommand(Msg);
                    }
                    else
                    {
                    Msg='MOVE HEAD RIGHT '+val+' DEGREES';
                    sendMessagecommand(Msg);
                       }
                       }
                    private function objectcommand(msg:String):void
                       {
                    var Msg3:String;
                       Msg3=msg+' OBJECT'+object.value;
                       sendMessagecommand(Msg3);
                       }

                       private function
updateTimer(evt:TimerEvent):void
                       {

                       sendMessagevideo('getcap0');
                       displayImage();

                       }

                       public function minimizeWindow():void
                 {

                 this.stage.nativeWindow.minimize();
                 }

                  public function maximizeWindow():void
                   {

               this.stage.nativeWindow.maximize();

                }

             public function restoreWindow():void

              {

               this.stage.nativeWindow.restore();
```

```
            }

            public function closeWindow():void
            {

            this.stage.nativeWindow.close();

            }

               ]]>

      </mx:Script>
      <mx:Button x="508" y="136" label="STOP" height="37" width="113"
id="emp" click="sendMessagecommand('STOP')" enabled="true"/>
      <mx:Button x="629" y="144" label="RIGHT" width="78"
click="sendMessagecommand('MOVE TURNING RIGHT 15 DEGREES')"/>
      <mx:Button x="431" y="144" label="LEFT"
click="sendMessagecommand('MOVE TURNING LEFT 15 DEGREES')"
width="69"/>
      <mx:Button x="516" y="100" label="FORWARD" width="96"   id="ad"
click="sendMessagecommand('MOVE WALKING FORWARD 1 STEP')"/>
      <mx:Button x="525" y="63" label="FAST F" width="76"
click="sendMessagecommand('MOVE WALKING FORWARD 5 STEPS')"/>
      <mx:Button x="516" y="189" label="BACKWARD" width="96"
click="sendMessagecommand('MOVE WALKING BACKWARD 1 STEP')"/>
      <mx:Button x="526" y="232" label="FAST B" width="76"
click="sendMessagecommand('MOVE WALKING BACKWARD 5 STEPS')"/>
      <mx:Button x="124" y="343" label="GOTO"
click="objectcommand('GOTO')"/>
      <mx:Button x="124" y="394" label="GRAB"
click="objectcommand('GRAB')"/>
      <mx:Button x="124" y="441" label="DROP"
click="objectcommand('DROP')" />
      <mx:TextArea id="chatMessage" editable="false" width="492"
height="166"  x="201" y="344"/>
      <mx:HSlider x="74" y="25" value="{Hsilder.value}" minimum="-60"
maximum="60" tickInterval="25" labels="[60,30,0,30,60]"
snapInterval="1" change="sendValue1(hslider.value)" id="hslider"
width="206"/>
      <mx:VSlider x="23" y="60" value="{Vsilder.value}" minimum="-45"
maximum="15" tickInterval="25" labels="[45,30,15,0,15]"
change="sendValue(hslider2.value)" id="hslider2" snapInterval="1"
height="194"/>
      <mx:NumericStepper x="10" y="291" id="Vsilder" minimum="-50"
maximum="50" stepSize="1" click="sendValue(Vsilder.value)"
width="50"/>
      <mx:NumericStepper x="312" y="21" id="Hsilder" minimum="-50"
maximum="50" stepSize="1" click="sendValue1(Hsilder.value)"/>
      <mx:TextInput x="201" y="314" id="imput" width="430"/>
      <mx:Button x="639" y="314" label="Send"
click="sendMessagecommand(imput.text)"/>
      <mx:Button label="_" click="minimizeWindow()" x="593"
width="34"/>
    <mx:Button label="Restore" click="restoreWindow()" x="525"/>
```

```
    <mx:Button click="maximizeWindow()" x="624" label="□" width="40"/>
    <mx:Button label="X" click="closeWindow()" x="660" width="33"/>
    <mx:Canvas x="66" y="63" width="{V4L_WIDTH}" height="{V4L_HEIGHT}"
id="canvas">
    </mx:Canvas>
    <mx:NumericStepper x="28" y="394" id="object" minimum="0"
maximum="10" stepSize="1"/>
    <mx:Label x="10" y="367" text="Object Number" fontSize="11"
color="#E0EAEC"/>


</mx:WindowedApplication>
```

ANEXO 2

PROTOCOLO COMUNICACIÓN

Sub-Protocol	Command	Response	Description
Connection	CONNECT<profile> <profile> | <Expert> <Specialize> <Standard>	OK WELCOME FROM<robot_name> KO UNKNOWN COMMAND <offending_command> <robot_name> | <RH-2> <ASIBOT> <HOAP3>	Request connection to a robot system according to the user type <profile>.
Control Negotiation	CONTROL<control_type>START CONTROL<control_type>STOP <control_type> | <Teleoperation> <Direct Command> <Autonomous> <Collaborative>	OK CONTROL <control_type> ACCEPTED KO CONTROL <control_type> REFUSED OK CONTROL <control_type> RELEASED KO UNKNOWN COMMAND <offending_command>	Give operator control of the robotic system according to the selected <control_type>.
Control Negotiation	CONTROL<control_mode>START CONTROL<control_mode>STOP <control_mode> | <Active mode> <Debug mode>	OK CONTROL <control_mode> ACCEPTED KO CONTROL <control_mode> REFUSED OK CONTROL <control_mode> RELEASED KO UNKNOWN COMMAND <offending_command>	Give operator control of the robotic system according to the selected <control_mode>.
Basic Movement	MOVE <movement_type><direction><count><unit of measure> <movement_type> | <Walking><forward:back>	OK COMMAND <command_id> <response> KO UNKNOWN COMMAND <offending_command>	Send order to move #<count> <units of

		<Turning>< left:right > <Head><up:down: left:right> <Link><linkID><coordinates> <Joint><jointID>< count>		<response>	<Started> <Completed> <Interrupted>	measure>. MOVE order can be send to the whole robot <Walking> & <Turning>. To an individual link <Link>. Or to a specific joint motor <Joint>.
Basic Movement	TOOL <tool_name> <action>	<tool_name>	<Hand><grasp_types…> <Gripper><close:open> < >< >	OK COMMAND <command_id> <response> KO UNKNOWN COMMAND <offending_command>	<response>	<Started> <Completed> <Interrupted>	Movements of the End-Effector tool. It could be a robot hand, a simple gripper or other.
Direct Command Execution	DIRECT <command>			OK COMMAND <command> <response> KO UNKNOWN COMMAND <offending_command>			Indicate that the action supplied as a parameter has to be executed immediately, skipping the input queue.
Configu	SET <parameter-name> <parameter-			OK <parameter-name> =			Set

ration	value> <parameter-name> | <step_lenght> <joint_limits> < > < >	<parameter-value> KO UNKNOWN COMMAND <parameter-name> KO VALUE OUT OF RANGE <parameter-value> KO OPERATION NOT ALLOWED TO <profile>	configuration values for various robot parameters like
Configuration	QUERY PARAM <parameter-name> <parameter-name> | <step_lenght> <joint_limits> < > < >	OK <parameter-name> = <parameter-value> KO UNKNOWN COMMAND <parameter-name> KO OPERATION NOT ALLOWED TO <profile>	Query the values of the robot configuration parameters.
Sensor reading	QUERY SENSOR [<label>, <label>, ... , <label>] <label> | <accelerometer> <gyroscope> <ZMP> <right_foot_force_sensor> <left_foot_force_sensor> <right_arm_force_sensor> <left_arm_force_sensor> <battery_level>	OK [<label>=<value>,<label>=< value>,...,<label>=< value>] KO UNKNOWN COMMAND <label> KO OPERATION NOT ALLOWED TO <profile>	Query the values from the robot sensor <label>.
Status	QUERY STATUS	OK STATUS = <robot_state> <robot_state> | <OK > <busy > <idle> <active> <engaged> <waiting>	Query the robot status.
Positioning	QUERY POSITION	POSITION <x-position> <y-position> <confidence>	Query the robot for

<table>
<tr><td></td><td></td><td></td><td>its location.</td></tr>
<tr><td>Goal setting</td><td>GOTO<location>
<table><tr><td><location></td><td><(<x><y>)
<OBJECT(<object_id>)>
< ></td></tr></table></td><td>OK COMMAND
<command_id>
<response>
KO UNKNOWN COMMAND
<offending_command>
<table><tr><td><response></td><td><Started>
<Completed>
<Interrupted></td></tr></table></td><td>Command robot to go to a define <location>; it could be a set of <x><y> coordinates, an object referential or other.</td></tr>
<tr><td>Goal setting</td><td>GRAB OBJECT <object_id></td><td>OK COMMAND
<command_id>
<response>
OK COMMAND
<command_id>
NEEDMOREINFO
KO UNKNOWN COMMAND
<offending_command>
<table><tr><td><response></td><td><Started>
<Completed>
<Interrupted></td></tr></table></td><td>Command robot to grab an object identified by <object_id>.</td></tr>
<tr><td>Goal setting</td><td>DROP OBJECT <object_id></td><td>OK COMMAND
<command_id>
<response>
OK COMMAND
<command_id>
NEEDMOREINFO
KO UNKNOWN COMMAND
<offending_command>
<table><tr><td><response></td><td><Started>
<Completed>
<Interrupted></td></tr></table></td><td>Command robot to drop an object identified by <object_id>.</td></tr>
</table>

Goal setting	USE OBJECT <object_id>	OK COMMAND <command_id> <response> OK COMMAND <command_id> NEEDMOREINFO KO UNKNOWN COMMAND <offending_command> | <response> | <Started> <Completed> <Interrupted> |	Command robot to perform a task with an object identified by <object_id>.
Goal setting	SELECT STRATEGYFOR <command_id> [<string>…<string>]	OK COMMAND <command_id> <response> KO UNKNOWN COMMAND <offending_command> | <response> | <Received> |	Robot request for user assistance in decision making for completion of a task.
Goal setting	USE STRATEGYFOR <command_id> <string>	OK COMMAND <command_id> <response> KO UNKNOWN COMMAND <offending_command> | <response> | <Received> <Started> <Completed> |	User command the selected strategy for the task.

Fuente:

Lorenzo BLasi, Deliverable D3.3: Design support software for ROBOT@CWE, 2008

Y Humanoid teleoperation system for space environments. Pierro, P.; Hernandez, D.; Gonzalez-Fierro, M.; Blasi, L.; Milani, A.; Balaguer, C.